

Courant Computer Science Report #16

October 1979

On the Complexity of the Satisfiability Problem

Allen T. Goldberg

Courant Institute of
Mathematical Sciences

Computer Science Department

New York University

Report No. NSO-16 prepared under Grant No.
NSF-MCS-76-24212 from the National Science Foundation



COURANT COMPUTER SCIENCE NOTES

- A101 ABRAHAMS, P. The PL/I Programming Language, 1979, 151 p.
- C66 COCKE, J. & SCHWARTZ, J. Programming Languages and Their Compilers, 1970, 767 p.
- D86 DAVIS, M. Computability, 1974, 248 p.
- M72 MANACHER, G. ESPL: A Low-Level Language in the Style of Algol, 1971, 496 p.
- M81 MULLISH, H. & GOLDSTEIN, M. A SETLB Primer, 1973, 201 p.
- S91 SCHWARTZ, J. On Programming: An Interim Report on the SETL Project.
Generalities; The SETL Language and Examples of Its Use. 1975, 675 p.
- S99 SHAW, P. GYVE — A Programming Language for Protection and Control in a
Concurrent Processing Environment, 1978, 668 p.
- S100 SHAW, P. " Vol. 2, 1979, 600 p.
- W78 WHITEHEAD, E.G., Jr. Combinatorial Algorithms, 1973, 104 p.

COURANT COMPUTER SCIENCE REPORTS

- 1 WARREN, H. Jr. ASL: A Proposed Variant of SETL, 1973, 326 p.
- 2 HOBBS, J. R. A Metalanguage for Expressing Grammatical Restrictions in Nodal
Spans Parsing of Natural Language, 1974, 266 p.
- 3 TENENBAUM, A. Type Determination for Very High Level Languages, 1974, 171 p.
- 4 OWENS, P. A Comprehensive Survey of Parsing Algorithms for Programming
Languages, ... 652+ p.
- 5 GEWIRTZ, W. Investigations in the Theory of Descriptive Complexity, 1974, 60 p.
- 6 MARKSTEIN, P. Operating System Specification Using Very High Level Dictions,
1975, 152 p.
- 7 GRISHMAN, R. (ed.) Directions in Artificial Intelligence: Natural Language
Processing, 1975, 107 p.
- 8 GRISHMAN, R. A Survey of Syntactic Analysis Procedures for Natural Language,
1975, 94 p.
- 9 WEIMAN, CARL Scene Analysis: A Survey, 1975, 62 p.
- 10 RUBIN, N. A Hierarchical Technique for Mechanical Theorem Proving and Its
Application to Programming Language Design, 1975, 172 p.
- 11 HOBBS, J.R. & ROSENSCHEIN, S.J. Making Computational Sense of Montague's
Intensional Logic, 1977, 41 p.
- 12 DAVIS, M. & SCHWARTZ, J. Correct-Program Technology/Extensibility of Verifiers,
with an Appendix by E. Deak, 1977, 146 p.
- 13 SEMENIUK, C. Groups with Solvable Word Problems, 1979, 77 p.
- 14 FABRI, J. Automatic Storage Optimization, 1979, 159 p. ..
15. LIU, S-C. & PAIGE, R. Data Structure Choice/Formal Differentiation.
Two Papers on Very High Level Program Optimization, 1979, 658 p.
- 16 GOLDBERG, A. T. On the Complexity of the Satisfiability Problem, 1979, 85 p.

Notes: Available from Department LN. Prices on request.

Reports: Available from Ms. Lenora Greene. Nos. 1,3,4,6,7,8,10 available in xerox only..

COURANT INSTITUTE OF MATHEMATICAL SCIENCES

251 Mercer Street

New York, New York 10012

COURANT INSTITUTE OF MATHEMATICAL SCIENCES

Computer Science

NSO-16

ON THE COMPLEXITY OF THE SATISFIABILITY PROBLEM

Allen T. Goldberg

October 1979

Report No. NSO-16 prepared under
Grant No. NSF-MCS-76-24212 from
the National Science Foundation

Table of Contents

I.	INTRODUCTION	1
	1.1 The Satisfiability Problem	1
	1.2 Complexity Theory	6
	1.3 Propositional Calculus	9
	1.4 Proof Systems for the Propositional Calculus	14
	1.5 Review of Previous Work on Expected Time Bounds	20
	1.6 Review of Previous Work on the Satisfiability Problem	22
	1.7 Summary of Results	24
II.	AVERAGE CASE ANALYSIS OF THE DAVIS-PUTNAM PROCEDURE	26
III.	PROOF SYSTEMS AND THE DAVIS-PUTNAM PROCEDURE	53
IV.	HORN SETS	68
	4.1 The Davis-Putnam Procedure and Horn Sets	68
	4.2 Proof Systems for Horn Sets	72
	4.3 Renaming	75
	REFERENCES	79

Acknowledgements

I would like to acknowledge the support and assistance of Martin Davis, Malcolm Harrison, Alan Konheim, Sandra Rapps, Salvatore Stolfo, Patricia Tobin, and the excellent typist Connie Engle. This thesis is dedicated to the memory of my father, Morris Goldberg, to whom I shall always be grateful.

ABSTRACT

An average case analysis of the Davis-Putnam procedure, an algorithm for testing propositional satisfiability, is presented. A variety of distributions on the set of problem instances is assumed. For each of these distributions a polynomial bound is obtained on the expected time complexity of the algorithm. When the uniform distribution is placed on the set of problem instances, the expected time of the Davis-Putnam procedure is shown to be $O(rn^2)$, where n is the number of clauses in the given set and r is the number of distinct atoms in the set. An investigation of the relationship of the Davis-Putnam procedure to resolution based procedures extends these results to resolution and to important restrictions of resolution such as the t -linear strategy. The power of the Davis-Putnam procedure is related to that of resolution-type proof procedures, where the length of refutations of unsatisfiable sets of clauses is taken as a measure of their power. The complexity of the satisfiability problem is examined for the special case of Horn sets. Questions related to transforming a set of clauses into a Horn set are resolved.

I. INTRODUCTION

1.1 The Satisfiability Problem

The problem of testing a formula of the propositional calculus in conjunctive normal form to determine whether or not it evaluates to *true* for some truth-value assignment to the variables is known as the *satisfiability problem*. This problem plays a fundamental role in many areas of Computer Science, especially computational complexity and theorem proving. It is intimately connected with a major open question in complexity theory — to determine whether there exist algorithms with polynomial time complexity for a class of problems known as the *NP*-complete problems. The satisfiability problem (SAT) was the first of many important combinatorial problems shown to be *NP*-complete [Cook, 1971; Karp, 1972]. These problems are equivalent in the sense that the existence of a polynomial time algorithm for any one of these problems implies that they all have such algorithms. SAT, because it has been the subject of extensive research, is a good representative problem from which to study this class. Logicians have examined the propositional calculus as a formal system for 100 years, commencing with Frege's *Begriffsschrift* published in 1879. These investigations can be usefully applied to help explore questions arising in complexity theory. One such open question is whether *NP* (the class of problems

computable nondeterministically in polynomial time) is closed under complementation. A positive answer to this question can be shown to be equivalent to the existence of a proof system for the propositional calculus in which the length of a proof of a formula is bounded by a polynomial in the length of that formula [Cook and Reckow, 1974]. This is a natural formulation of the closure question that can take advantage of the extensive study proof systems have undergone.

SAT is important not only as an avenue for studying NP-completeness, but also because of its role as a foundation for logistic systems. Any decision or semidecision procedure for a logical theory (based on classical logic) must incorporate within it a solution to SAT. Thus, lower bounds on the complexity of SAT apply to any logical theory. In many cases this embedding appears explicitly as part of the decision or semidecision procedure. When this occurs one has the freedom to choose the best known algorithm for SAT. This gives a practical imperative for the development of effective algorithms for SAT. We shall demonstrate that the Davis-Putnam Procedure [Davis and Putnam, 1960; Davis, Loveland and Logemann, 1962] is an effective algorithm for SAT. Decision procedures based on the method of elimination of quantifiers, which transform a formula to an equivalent one in the propositional calculus, can use algorithms which are as efficient as the Davis-Putnam Procedure. Herbrand's theorem

is the basis of semidecision procedures for first order logic in which instantiations of the clauses are tested for propositional satisfiability. Some theorem provers of this type, notably the linked-conjunct method [Davis, 1963; Yarmush, 1976] and a similar one described by Schwartz [Schwartz, 1978] test propositional satisfiability explicitly.

SAT is so fundamental that diverse problems in all areas of Computer Science embed SAT in their algorithmic solutions. This may occur, for example, in the processing of data base queries or in digital circuit design.

The question of whether there is a polynomial time algorithm for testing satisfiability (i.e. $P = ? NP$) is a fundamental open question of complexity theory. The prevalent view is that $P \neq NP$ so that there are no polynomial time algorithms for any of the NP -complete problems. The evidence for this conjecture is the failure of many researchers to find a polynomial time algorithm for an NP -complete problem, despite the attention given to this broad class of problems. On the other hand, no significant lower bounds have been proved for any NP -complete problem. A proof that $P \neq NP$ does not appear imminent.

From the perspective of complexity theory, SAT is considered to be intractable by presently available algorithms. Intractability is usually equated with exponential worst case time complexity. Practitioners, while finding these considerations useful, continue to attempt to solve problems

considered intractable whenever the advantages of obtaining a solution outweigh the computational cost. The work that has been done in mechanical theorem proving has these characteristics. In these practical contexts average case analysis, based upon the actual distribution of problem instances encountered, is of greater interest than the worst case analysis. Unfortunately, good average case results are difficult to obtain because properties of the actual distribution may be unknown, and the technical problems encountered in performing such an analysis may be formidable. Worst case analyses give upper bounds on the expected time complexity of an algorithm and so conservatively estimate the algorithm's actual performance. If these conservative bounds guarantee acceptable performance, or if they are close to a known lower bound, then these bounds are particularly informative. However, an algorithm should not be rejected on the basis of a disappointing worst case analysis. Under these circumstances, an average case analysis may indicate that the algorithm will almost always meet desired performance criteria. The quicksort algorithm [Aho, Hopcroft and Ullman, 1976] which has an $O(n^2)$ upper bound on its worst case performance is considered a more effective algorithm than many $O(n \log n)$ sorting algorithms. Quicksort is $O(n \log n)$ on the average, assuming each permutation of the keys is equally likely. The simplex method for solving linear programming problems also exhibits this

phenomenon. Its worst case analysis yields an exponential bound [Klee and Minty, 1972], yet acceptable performance has been observed in practice [Karp, 1976].

It appears that this is the case for the satisfiability problem as well. Namely, when the Davis-Putnam procedure is used to solve the problem in a practical context, the observed behavior of the algorithm is such that the problem is tractable. However, with some difficulty, exponentially difficult cases can be constructed [Tseitin, 1968]. In this dissertation, an average case analysis of the Davis-Putnam procedure is presented that supports the tractability of this NP -complete problem. A family of results is obtained that give polynomial bounds for various distributions on the problem instances. In particular, if the uniform distribution is placed on the sample space of problem instances, then the expected time complexity of the Davis-Putnam procedure is shown to be $O(rn^2)$, where r is the number of atoms appearing in the given set of clauses and n is the number of clauses in the set. It shall also be argued that these favorable results are due to an essential feature of the Davis-Putnam procedure and not to favorable distributional assumptions. It is also a consequence of these results that instances on which the algorithm does take exponential time are very rare.

1.2 Complexity Theory

Problems, like SAT, whose output is either *yes* (the set is satisfiable) or *no* (it is not) are considered, in the framework of complexity theory, to be language accepting problems. An *alphabet*, Σ , is a finite, nonempty set of distinct symbols. Σ^* , the set of *words* over Σ , consists of all finite sequences of symbols of Σ . A *language*, L , is any subset of Σ^* . The language accepting problem for L is "given $w \in \Sigma^*$, is $w \in L$?". If we take Σ to be the set of all symbols needed to specify formulas of the propositional calculus in conjunctive normal form, then we can let L be the set of all satisfiable formulas. We are particularly interested in languages that are *recursive*, i.e. such that there is an algorithm, AL , which takes a string $w \in \Sigma^*$ and eventually halts indicating $w \in L$ or $w \notin L$. There are many underlying models which give precise meaning to the notion of an algorithm. We shall assume deterministic Turing machines as our computational model. The results obtained in this thesis apply to any reasonable known model of deterministic computation. See [Garey and Johnson, 1979] for details. We shall also consider nondeterministic computations and use nondeterministic Turing machines as our computational model. For $w \in \Sigma^*$, let $|w|$ denote the length of w . Let $\Sigma^n = \{w \in \Sigma^* \mid |w| = n\}$. Let AL be an algorithm that accepts L . The *worst case time complexity* of AL is a function $T_{\text{WORST}, AL}(n)$ defined as

$$T_{\text{WORST,AL}}(n) = \max_{|w|=n} [\text{number of steps AL performs given input } w].$$

Here we are measuring the complexity as a function of the most difficult instance of a given size. An alternative is to consider an average over all instances of a given size. This requires that a probability distribution function P_n be placed on Σ^n , for each n . This function should reflect the probability that a particular instance appears when solving the problem in some given context. Given such distribution functions P_n

$$T_{\text{AVERAGE,AL}}(n) = \sum_{|w|=n} P_n\{w\} [\text{number of steps AL performs given input } w]$$

(Without ambiguity the subscript on the distribution function will be dropped.)

A problem which is solved by an algorithm whose worst case time complexity is bounded by a polynomial in the length of the input is considered tractable. If no such algorithm exists it is considered intractable. Thus, tractability is considered equivalent to membership in the class

$P = \{L \mid \text{there is an algorithm, AL, accepting } L \text{ such that}$

$T_{\text{WORST,AL}}(n) \leq cn^k \text{ for some constants } c \text{ and } k\}.$

$NP = \{L \mid \text{there is a nondeterministic algorithm, NAL,}$

accepting L such that $T_{\text{WORST,NAL}}(n) \leq cn^k$ for some constants

c and $k\}$. It is understood that a nondeterministic

algorithm accepts L if there is some accepting computation

sequence accepting w just in case $w \in L$. Furthermore, the

number of steps taken by the shortest such sequence is used

to measure the time complexity of the algorithm on w . Because many diverse and fundamental problems including SAT are in NP , the question of whether $P = NP$ is basic. In 1971, Cook [Cook, 1971] showed that SAT was the "hardest" problem in NP . That is, if SAT is in P then $P = NP$. This was proved using the notions of reducibilities and completeness that were developed in recursion theory. If L and L' are two languages over Σ then L is *p-reducible* to L' if there is a function $f: \Sigma^* \rightarrow \Sigma^*$, computable by a deterministic algorithm in polynomial time such that

$$w \in L \text{ iff } f(w) \in L' .$$

Clearly, if L is *p-reducible* to L' and L' is in P then L is in P as well. L is *NP-complete* if L is in NP and for any L' in NP , L' is *p-reducible* to L . Cook showed that SAT is *NP-complete*. This having been established, any other problem in NP can be shown to be *NP-complete* by proving that SAT is *p-reducible* to it. Karp [Karp, 1972] showed that in fact most problems known at the time to be in NP but not known to be in P are *NP-complete*. In fact, although many problems have been added to the list of *NP-complete* problems (we shall add another), few have been added to the three problems in NP that could not and still have not been shown to be either in P or *NP-complete*. (They are GRAPH ISOMORPHISM, COMPOSITE NUMBERS and LINEAR PROGRAMMING [Garey and Johnson, 1979]).

1.3 Propositional Calculus

There are many formal systems that are properly called the propositional calculus. They each specify a formal language in which *formulas* are inductively built from a countable set of *atoms* or *propositional variables* (denoted $p, q, x_1, x_2, \dots$), using *connectives* such as $\&$ (and), $\vee$ (or), $\rightarrow$ (implies), $\neg$ (negation) in a manner which guarantees that the formulas are *well formed*. An *interpretation, valuation* or *assignment to the propositional variables* is a function from the atoms to the set of truth values, $B = \{\text{True}, \text{False}\}$. The *truth value of a formula* with respect to an interpretation is computed inductively on the structure of the formula. Connectives are assigned their standard meanings as Boolean functions of the proper arity. Thus $\&$ is interpreted as a function from $B^2 \rightarrow B$ whose value is True iff both its arguments are True. When meanings have been assigned to the connectives and an assignment to the propositional variables is made, the truth value of a formula can be computed. The choice of connectives varies with the formulation of the logic. However, all Boolean functions should be realizable by a formula of the language. From the point of view of complexity theory the choice of connectives has an importance beyond being able to realize all functions. Some languages can realize Boolean functions with shorter formulas than others. This, together with the choice of rules of inference and axioms, affect the power of logistic systems to prove tautologies concisely.

The language attracting our greatest attention will utilize the three connectives $\&$, $\vee$, and $\neg$. Any formula of this language is equivalent (i.e. realizes the same Boolean function) as a formula in *conjunctive normal form* (CNF). A formula is in CNF if it is a conjunction, $C_1 \& C_2 \& \dots \& C_n$, in which each *conjunct* or *clause*, C_i , $i = 1, \dots, n$, is a disjunction of *literals*, $L_{i1} \vee L_{i2} \vee \dots \vee L_{in_i}$, where each L_{ij} is an atom or a negation of an atom. Many proof systems operate exclusively on CNF formulas. Formulas in CNF are frequently written as sets whose members are the formulas' clauses which are themselves written as sets or strings of literals. This representation is unambiguous with respect to equivalence of formulas because both conjunction and disjunction are commutative and associative operations. A literal is *positive* if it is an atom, *negative* otherwise. The *complement* of a positive (negative) literal, L , is the negative (positive) literal, $\bar{L}$, with the same atom. We shall assume that a clause does not contain both a literal and its complement. (A formula obtained by dropping such a clause is equivalent to the original formula.) A formula is *satisfiable* if there is an interpretation for which the formula is true. A formula is a *tautology* if it is true for every interpretation. The *satisfiability problem* is to identify CNF formulas that are satisfiable.

A formula which is a disjunction of conjunctions of

literals is in *disjunctive normal form* (DNF). The *tautology problem* is to identify DNF formulas that are tautologies. The tautology problem and satisfiability problems are duals. If ϕ is a formula in DNF, then ϕ is a tautology iff $\bar{\phi}$ (which can easily be put in CNF by interchanging $\&$'s and $\vee$'s and replacing literals by their complements) is unsatisfiable. Decision procedures which test a DNF formula for being a tautology are *affirmation* procedures; those testing a CNF formula for satisfiability are *refutation* procedures.

The Davis-Putnam procedure (DPP) is a refutation procedure to decide SAT. Given a set of clauses $S = \{C_1, C_2, \dots, C_n\}$ the procedure applies one of three different rules. Each application of a rule (a step of the procedure) reduces the problem of satisfiability of S to the satisfiability of one or possibly two derived sets, each set containing fewer literals than the original. The *pure literal rule* can be applied if there is a literal L which appears in a clause of S while its complement, $\bar{L}$, does not. Let $S_1 = \{C \mid C \in S \text{ and } L \notin C\}$. Then, S_1 is satisfiable iff S is satisfiable. For, let M be an interpretation that satisfies S . Since $S_1 \subset S$, M satisfies S_1 as well. Conversely, let M satisfy S_1 . Since neither L nor $\bar{L}$ appear in S_1 , it is permitted to assume that M assigns True to L . Then M will satisfy S .

If there is a clause consisting of only a single literal, L , (a *unit clause*) then the *unit clause rule* may

be applied. Let $S_1 = \{C - \{\bar{L}\} \mid C \in S \text{ and } L \notin C\}$. Again, S is satisfiable iff S_1 is satisfiable. For, any interpretation which satisfies S must assign True to L and, therefore, $\bar{L}$ is assigned False. Thus for any clause $C \in S$, M satisfies $C - \{\bar{L}\}$ and so satisfies S_1 . Conversely, let M satisfy S_1 . Since L and $\bar{L}$ do not appear in S_1 assume M assigns True to L . M satisfies S . Finally, the *splitting rule* is applied whenever neither of the two other rules are applicable. Choose any literal L . Let $S_1 = \{C - \{\bar{L}\} \mid C \in S \text{ and } L \notin C\}$ and $S_2 = \{C - \{L\} \mid C \in S \text{ and } \bar{L} \notin C\}$. Then, S is satisfiable iff S_1 or S_2 is satisfiable. For, suppose S is satisfied by an interpretation M . Either L or $\bar{L}$ is assigned True by M . Then either S_1 or S_2 is satisfied by M . Conversely, assume, without loss of generality, that S_1 is satisfied by M . Since L and $\bar{L}$ do not appear in S_1 , we can assume that M assigns True to L and so satisfies S . The DPP applies a rule at each step reducing the satisfiability of S to the satisfiability of S_1 and possibly S_2 . It then recursively applies itself to these problems. Since at each step the desired sets contain fewer distinct literals and fewer clauses, the process will eventually terminate. If at any stage a set has a clause containing no literals, i.e., the *empty clause*, then that set is unsatisfiable. The empty set of clauses is vacuously satisfiable. Because the unit and pure literal rules reduce the satisfiability of S to that of the single set S_1 , they

are the more desirable rules to apply. An Algol-like description of the DPP follows.

```
procedure DPP(S);  
1.  if S =  $\emptyset$  then return satisfiable;  end if;  
  
2.  if  $\emptyset \in S$  then return unsatisfiable;  end if;  
  
3.  if the unit rule applies to S then  
    if DPP(S1) = satisfiable  
        then return satisfiable;  
    else return unsatisfiable;  
    end if;  
end if;  
4.  if the pure literal rule applies to S then  
    if DPP(S1) = satisfiable  
        then return satisfiable;  
    else return unsatisfiable;  
    end if;  
5.  else (apply the splitting rule) if DPP(S1)=satisfiable  
    or DPP(S2) = satisfiable  
        then return satisfiable;  
    else return unsatisfiable;  
    end if;  
end if;  
end DPP;
```

1.4 Proof Systems for the Propositional Calculus

Traditionally a proof system is a formalism used to help identify the valid formulas of a logistic theory. It consists of a set of axioms and rules of inference. A *proof* of a formula ϕ is a sequence of formulas, ending with ϕ , in which each formula of the sequence is an axiom or is inferred by some rule of inference applied to formulas preceding it in the sequence. If a formula ϕ has a proof it is called a *theorem* of the proof system. If a proof system is to accomplish its task of identifying valid formulas it must be *sound*, that is, only valid formulas should be provable. Furthermore, it must be possible to decide algorithmically when a sequence of formulas is a proof. In fact this algorithmic verification should be computationally fast. It is desirable, but not always attainable, that a proof system be *complete*, that is that every valid formula should have a proof. We shall adopt a definition of a proof system, due to Cook [Cook and Reckow, 1974] which emphasizes the notion of proof as an object which verifies the validity of a formula. This definition will capture many of the important properties of the notion of a proof yet admit as proof systems a broader range of objects than those based on axioms and rules of inference. A *proof system* for the propositional calculus is a function F from a subset of Σ^* (Σ is a finite alphabet, not necessarily the alphabet of the language of the propositional calculus) onto the set of valid

formulas of the propositional calculus, which can be computed in polynomial time. If $F(w) = \phi$ then w is a proof of ϕ . Since the range of F is only valid formulas, F as a proof system is sound. F is complete since it is onto. Since F is computed in polynomial time, the verification that w is a proof of ϕ is computationally efficient. Refutation procedures fall into this framework by regarding a refutation of $\bar{\phi}$ as a proof of ϕ . For example, the DPP may be regarded as a proof system where the proof of a set of clauses consists of a trace of the computation performed by the algorithm when given that set. This trace may be the tree of derived sets generated by the procedure. Sufficient detail must be given so that verification that the string presented is actually the trace of a DPP refutation can be carried out algorithmically in polynomial time.

A proof system, F , is *polynomially bounded* if there exists a polynomial p such that for any tautology ϕ , there is a proof w of ϕ satisfying $|w| \leq p(|\phi|)$. The definition of proof system allows the following theorem to be easily proved.

Theorem (Cook). NP is closed under complementation iff there exists a polynomially bounded proof system for the propositional calculus.

Let F_1 and F_2 be two proof systems for the same formulation of the validity problem for the propositional calculus (for example, they both attempt to decide CNF satisfiability).

F_1 *polynomially simulates* (p-simulates) F_2 if there exists a function f which maps a proof in F_2 of ϕ into a proof in F_1 in polynomial time. Thus proofs in F_2 can be turned into proofs in F_1 with only a polynomial increase in the length of the derivation. Thus if F_1 is not a polynomially bounded verification system neither is F_2 . [Reckow, 1976] has compared the power of many classical proof systems using this ordering. Our concern will be focused on resolution related proof systems.

Resolution is a refutation procedure which operates on formulas in CNF. Basic to the procedure is the *cut* or *resolution* rule. Given 2 clauses, called *parent clauses*, $L \vee L_1 \vee \dots \vee L_k$ and $\bar{L} \vee L'_1 \vee L'_2 \vee \dots \vee L'_k$, infer the clause $L_1 \vee L_2 \vee \dots \vee L_k \vee L'_1 \vee L'_2 \vee \dots \vee L'_k$, as an additional clause in the conjunction. The complementary pair of literals L and $\bar{L}$ are called the *annihilated* literals. The derived clause is called the *resolvent*. When the unit clause L is resolved with its complement $\bar{L}$, the empty clause, denoted $\square$, is the resolvent. Since the empty clause is unsatisfiable, any set of clauses which can derive the empty clause is unsatisfiable. Resolution is sound and complete. Given a set of clauses, S , $S \cup \{\text{resolvents of clauses in } S\}$ is equivalent to S . Furthermore, if S is unsatisfiable, then the empty clause is derivable from S . The resolution rule can be incorporated into a decision procedure testing the satisfiability of S .

repeat forever;

if $\square \in S$ then S is unsatisfiable.

Find a pair of clauses in S containing complementary literals, whose resolvent is not in S . If no such pair exists, S is satisfiable. Otherwise add the resolvent to S .

end repeat;

The length of the resolution proof is the number of clauses generated by the resolution rule necessary for the derivation of the empty clause. The above procedure may generate many extraneous resolvents. There are various ways to display the proof or refutation. One way is to enumerate a sequence of clauses, each element of the sequence being either a clause from the given set or the resolvent of previous clauses in the sequence. The final clause of the sequence is the empty clause. Alternatively, these clauses can be labels of nodes of a directed acyclic graph (DAG). In such a graph a directed edge is placed from clause C_1 to C_2 iff C_2 is obtained by resolution, with C_1 being one of its parents. These two representations are nearly identical; the DAG representation explicitly indicates the parents of a resolvent, making verification of the correctness easier. However, since recovering the parents of a resolvent is computationally easy, the difference between the DAG and linear representation is not essential. In a third variation, derivations are represented as trees. In such a representation, a clause can be the parent of at most one

derived clause. Thus, if a clause is used more than once in a derivation, the subtree deriving it must be repeated. This transformation can cause an exponential blow-up in the number of nodes needed to represent a derivation as a tree instead of a DAG. Tseitin showed that this blow-up is unavoidable by proving that tree resolution cannot p-simulate resolution proofs presented as DAGs [Tseitin, 1968].

Regular resolution is a restricted form of resolution. A resolution derivation represented as a DAG is *regular* if along any path from the root of the DAG to a leaf there is no atom that is annihilated twice. Tseitin [Tseitin, 1968] showed that regular resolution is not a polynomially bounded proof system. In the obvious way, the regularity condition can be imposed on tree resolution in which case we speak of *regular tree resolution*. For tree resolution, the regularity condition is easily fulfilled. Loveland and Tseitin have each shown that without increasing the length of the proof a tree resolution proof can be made regular [Tseitin, 1968, Loveland, 1970].

There are many other resolution based proof systems whose origins can be found in the theorem proving literature of the late 1960's [Chang and Lee, 1973]. In 1965, Robinson recognized that the resolution rule, in the context of first order logic, is particularly amenable to machine implementation [Robinson, 1965]. There is a simplicity achieved in

having only a single rule of inference. More important, the number of possible resolutions that can be performed at any stage is manageable, usually being somewhat less than ten [Schwartz, 1978]. Future work decreased this number further by specifying conditions that must be met before a resolution is performed. The thrust of these so-called restriction strategies was to prune the search space of possible refutations, yet still maintain the completeness of the proof system. Unfortunately, many of these strategies were apt to prune successful as well as unsuccessful branches indiscriminately. However, some of the strategies, notably linear resolution, do not substantially increase the lengths of proofs as compared with those obtainable by regular resolution [Kowalski and Kuehner, 1971].

Proof systems are a valuable tool for the investigation of the $NP = \overline{NP}$ question, but do not directly aid the development of good algorithms for SAT. Finding a powerful proof system is equivalent to finding a good nondeterministic algorithm for SAT. At each step of its computation the nondeterministic algorithm guesses the correct rule of inference to apply. A good algorithm for SAT could be based on the proof system if the level of nondeterminism is not too high. Reckow's study of proof systems [Reckow, 1976] puts Frege systems (systems like P_1 [Church, 1956], and others familiar from logic texts) as the most powerful, natural deduction systems next, followed by resolution systems

and truth tables. As we go from the weakest to the strongest systems the nondeterminism increases. Thus, the choice of a proof system on which to base an algorithm for SAT involves a tradeoff between length of proof and nondeterminism.

1.5 Review of Previous Work on Expected Time Bounds

The unacceptable worst case performance of an algorithm can be the result of a few rare instances on which the algorithm performs poorly. Work by Karp [Karp, 1976] suggests that this, in fact, may be the case for many *NP*-complete problems. He reports results in which polynomial time algorithms construct near optimal solutions on nearly all problem instances. For example, in $O(n \log n)$ time, almost all traveling salesman problems can be solved to within $1 + \epsilon$ of an optimal solution. Results regarding approximation algorithms may not carry over to exact algorithms, but do encourage the search for exact algorithms for *NP*-complete problems having good average case performance.

Among the first algorithms to be analyzed for average case complexity were sorting and searching algorithms. A thorough presentation can be found in Knuth [Knuth, 1973].

Erdős and Renyi introduce the notation of a random graph [Erdős and Renyi, 1959]. Using this model, many graph theoretic algorithms have been analyzed [Karp, 1976,

Grimmet and McDiarmid, 1975; Spira, 1973; Posa, 1976; Perl, 1977]. Complexity results have been so favorable using this model [Karp, 1976], that the relevance of the model has been called into question.

Recently, diverse combinatorial problems have been the subject of expected time analyses. Yao and Yao [Yao and Yao, 1978] looked at the problem of selecting the k^{th} largest key, assuming all $n!$ orderings equally likely. [Doyle and Rivest, 1976; Knuth and Schonhage, 1977] have analyzed two important union-find algorithms, showing their expected linearity. [Bentley and Shamos, 1978] give a linear expected time algorithm to find the convex hull of a set of points in Euclidean space. They emphasize the "divide and conquer" nature of the algorithm and the very weak assumptions made on the distribution function. Further work includes [Floyd and Rivest, 1975] on selection, [Guibas and Szemendi, 1976] on hashing and [Yao, 1977] on pattern matching.

1.6 Review of Previous Work on the Satisfiability Problem

Cook's theorem has established the framework from which all work on the complexity of SAT is based. In a positive direction, the search continues for finding better algorithms, proof procedures and approximation schemes. In a paper entitled "Toward Feasible Solutions to the Tautology Problem," Dunham and Wang [Dunham and Wang, 1974] present a variety of algorithms which yield efficient solutions for cases which defeat the well known algorithms. In fact it is difficult to find a class of instances for which none of the methods considered will work.

A popular method of coping with NP-completeness is to consider approximation algorithms which give near optimal solutions in polynomial time (for a bibliography see [Garey and Johnson, 1976]). This approach is useful for optimization problems, like the traveling salesman problem, in which the goal is to maximize an objective function. For "yes/no" problems like SAT the notion is not as useful. However, there exists an approximation algorithm for the maximum satisfiability problem — find the assignment which satisfies the maximum number of clauses in the set [Johnson, 1974]. The algorithm described does not give good approximations unless the number of literals appearing in each clause is relatively large. [Lieberherr and Spector, 1978] prove interesting results regarding maximum satisfiability, namely, if a set of clauses does not contain a pair of

complementary unit clauses, then there is an interpretation obtainable in polynomial time which satisfies at least hn clauses, where h is the golden ratio $((\sqrt{5} - 1)/2 \approx 0.618)$, and n is the number of clauses. Furthermore, h is maximal, that is, for any larger h the problem of finding such an interpretation is *NP*-complete.

Negative results on SAT give exponential lower bounds for algorithms based on particular proof procedures. The most important result of this type is due to Tseitin [Tseitin, 1968], who proved that regular resolution is an exponential proof system. [Galil, 1977] improved the result slightly by increasing the lower bound on the length of regular resolution proofs from $O(2^{\sqrt{n}})$ to $O(2^n)$. [Kozen, 1977] has applied the same idea of proving lower bounds for natural proof systems for other logical theories.

Many variations on SAT have been examined. The 3-satisfiability problem, in which clauses are restricted in length to at most 3 literals, is *NP*-complete [Cook, 1971], whereas 2-satisfiability is polynomial. For interesting generalizations and problems related to SAT see [Shaefer 1978; Garey, Johnson and Stockmeyer, 1976; Ladner, 1977].

1.7 Summary of Results

In Chapter II, an average case analysis of the DPP is presented. Conditions for a probability distribution function on problem instances are stated which assure that the expected time of the procedure is polynomial. The variance of the algorithm is discussed. It is shown that instances on which the DPP requires exponential time are rare.

In Chapter III results are presented which relate the DPP to prior work on proof systems for the propositional calculus. A variant of the DPP (referred to as DPP"), which is the historical predecessor of the DPP, is described. DPP" has been studied in this context because it is easily regarded as a particular type of resolution procedure. It is shown that a structural isomorphism exists between DPP and regular tree resolution. Using this characterization of the DPP proof system in terms of resolution, DPP and DPP" are compared. It is found that they are incomparable with respect to the p-simulate relation on proof systems. Thus, despite surface similarities, fundamental differences exist between DPP and DPP". It is then shown that refutation graphs [Shostack, 1976] can be quickly constructed from DPP refutations. From a refutation graph other types of resolution proofs, such as t-linear refutations, can be efficiently obtained. Thus, the results of Chapter II extend to many of the variants of resolution

discussed in the theorem proving literature.

In Chapter IV, Horn sets and their relation to the DPP and resolution is discussed. The DPP, modified for this special case, can decide satisfiability in linear time. Without modification the DPP is efficient but not linear. It is shown that the resolution refutation that can be generated from the DPP using the results of Chapter III is an input refutation. It is the shortest possible resolution proof, consisting of only as many resolutions as there are distinct atoms in the set. The DPP" generates unit refutations for Horn sets. These refutations are generally longer than the input refutations but are still linear in the size of the set.

The final section of the chapter discusses renamings, a satisfiability preserving transformation which can sometimes transform a set into a Horn set. It is shown that the question of whether a set is renamable as a Horn set is decidable in polynomial time, but the problem of maximizing, by renaming, a suitably defined notion of degree of "Horn-ness" is *NP*-complete.

These results can be taken as strong evidence that despite the *NP*-completeness of SAT, the problem is, in practice, tractable. Results like these should be sought for other *NP*-complete problems.

II. AVERAGE CASE ANALYSIS OF THE DAVIS-PUTNAM PROCEDURE

"Divide and conquer" [Aho, Hopcroft and Ullman, 1974] has proved to be a useful technique in the design of efficient algorithms. A "divide and conquer" algorithm, AL, solves a problem, S, of size n , by dividing S into smaller problems; it recursively applies AL to the smaller problems and uses the partial solutions to construct the solution of S. If the time needed to divide the problem is small (for example, linear) and significantly, if the number and size of these subproblems is also small, then an efficient algorithm will result. The DPP is a "divide and conquer" algorithm which in linear time divides a set of clauses into at most two smaller sets. However, these sets in some cases will not be of substantially smaller size than the original. As a result, the DPP has an exponential worst case time complexity. Nevertheless, as will be seen, on the average, the subproblems are small enough to assure that the *expected* time complexity of the DPP is polynomial.

The usual *modus operandi* of complexity analysis is to parameterize instances of a problem (see Chapter I) by the length of the description of the problem presented to the algorithm. The complexity of the algorithm is measured as a function of the length of this description. For example,

the complexity of an algorithm to test primality of positive integers is measured as a function of the number of digits of the number. The length of the description of a problem instance is a crude, problem independent measure of the difficulty of the instance relative to the other problem instances. Better results can frequently be obtained by more appropriate parameterizations. The complexity of graph algorithms is better described as a function of the number of nodes and edges in the graph. In a similar fashion, we shall parameterize instances of the satisfiability problem using two parameters: r , the number of atoms in the atom set from which the clauses are chosen, and n , the number of clauses in the set.

Let $C(r)$ be the set of all clauses that can be constructed from the atom set $A = \{x_1, x_2, \dots, x_r\}$. There are 3^r clauses in $C(r)$. This is because the definition of clause excludes the simultaneous appearance of a literal and its complement. Thus, for each of the r atoms, the atom may appear in a clause as a positive literal, negative literal or not at all, generating 3^r different clauses.

Let $K(n, r) = \prod_{i=1}^n C(r)$. That is, an element of $K(n, r)$ is a sequence of n clauses chosen from the set $C(r)$. There are technical advantages to defining $K(n, r)$ as a sequence of clauses rather than as a set of clauses. The clauses are ordered, allowing for the appearance of identical clauses. Furthermore, there is a natural mechanism that extends a

probability density function (pdf) from $\mathcal{C}(r)$ to $K(n,r)$, as the n -fold product space of $\mathcal{C}(r)$. The process in which $S \in K(n,r)$ is chosen randomly according to this induced pdf can be envisioned as the sequence formed by making n independent choices of clauses from $\mathcal{C}(r)$ according to the distribution placed on $\mathcal{C}(r)$. Thus, if $P_{\mathcal{C}(r)}\{C_i\}$, $C_i \in \mathcal{C}(r)$, $i=1, \dots, n$ is the probability of choosing C_i from $\mathcal{C}(r)$, then the probability of choosing the sequence $\langle C_1, \dots, C_n \rangle$ from $K(n,r)$ is $P\{\langle C_1, \dots, C_n \rangle\} = \prod_{i=1}^n P_{\mathcal{C}(r)}\{C_i\}$. (Nevertheless for the sake of ease of expression, $S \in K(n,r)$ will sometimes be referred to as a *set* of n clauses.) Let $p_{i,r} = P_{\mathcal{C}(r)}\{C \mid C \in \mathcal{C}(r) \text{ and } x_i \in C\}$, $i = 1, \dots, r$, and $q_{i,r} = P_{\mathcal{C}(r)}\{C \mid C \in \mathcal{C}(r) \text{ and } \bar{x}_i \in C\}$, $i = 1, \dots, r$.

In the absence of any knowledge regarding the relative frequency of problem instances and of the underlying theoretical distribution, average case analyses assume that all instances are equally likely. Hence the analysis of the DPP will proceed by first assuming that the uniform distribution has been placed on $\mathcal{C}(r)$ and, consequently, on $K(n,r)$.

Lemma 2.1. If the uniform distribution has been placed on $\mathcal{C}(r)$, then $p_{i,r} = q_{i,r} = 1/3$, $i = 1, \dots, r$.

Proof. For any atom x_i , exactly $1/3$ of the clauses in $\mathcal{C}(r)$ contain x_i , and exactly $1/3$ contain $\bar{x}_i$. □

It might be expected that obtaining good bounds on the complexity of the DPP would be contingent upon the frequent applicability of the unit and pure literal rules, since it is the splitting rule which introduces the possibility of exponentially long computations. But this is not the case. The pure literal and unit rules can be regarded as special cases of the splitting rule. Let the unit rule be applicable to a set of clauses S and let $\{L\}$ be a unit clause in S . If we split S into sets

$$S_1 = \{C - \{\bar{L}\} \mid C \in S \text{ and } L \notin C\} \text{ and}$$

$$S_2 = \{C - \{L\} \mid C \in S \text{ and } \bar{L} \notin C\}, \text{ then } \emptyset \in S_2,$$

and the DPP would immediately report S_2 unsatisfiable. Thus application of the unit rule on literal L is not essentially different from application of the splitting rule. Next, let L be a literal that appears pure in S , i.e., $C \cap \{\bar{L}\} = \emptyset$ for every $C \in S$. In this case, application of the splitting rule on L creates sets S_1 and S_2 with $S_1 \subseteq S_2$. Thus the pure literal rule only considers the satisfiability of S_1 . Therefore the point of view taken here is that for the DPP, the splitting rule is fundamental. The other two rules amount to a preference for splitting on certain literals and a useful optimization — detecting the relation $S_1 \subseteq S_2$. Improved performance of the DPP results from minimizing the sizes of the sets S_1 and S_2 . In effect, the unit and pure literal rules are instances of the application of this principle.

We shall analyze a procedure DPP' which only applies the splitting rule. Furthermore, DPP' does not attempt to make a strategic choice of the literal on which the set is split. It instead chooses the literal at random giving equal probability to each literal in the literal set. The algorithm for DPP' follows:

procedure DPP'(S);

1. *if* $S = \emptyset$ *then return* satisfiable *end if*;
2. *if* $\emptyset \in S$ *then return* unsatisfiable *end if*;
3. Let L be an arbitrary literal from L.

$S_1 := \{C - \{\bar{L}\} \mid C \in S \text{ and } L \notin C\};$

$S_2 := \{C - \{L\} \mid C \in S \text{ and } \bar{L} \notin C\};$

if DPP'(S₁) = unsatisfiable *and*

DPP'(S₂) = unsatisfiable

then return unsatisfiable;

else return satisfiable; *end if*;

end DPP';

Lemma 2.2. Let S be a set of clauses chosen uniformly from $\mathcal{C}(r)$. Let S₁ and S₂ be defined as above. Then the clauses of S₁ and S₂ are each uniformly distributed over $\mathcal{C}(r - 1)$.

Proof. For each clause $C \in \mathcal{C}(r - 1)$, there are precisely two clauses, namely C and $C \cup \{L\}$ of $\mathcal{C}(r)$ whose presence in S will result in the occurrence of C in S₁. So the probability

that C is in S_1 is equal to the probability that either C or $C \cup \{L\}$ is in S . Since clauses in S are chosen uniformly from $\mathcal{C}(r)$, for any $C_1, C_2 \in \mathcal{C}(r-1)$,
 $P\{C_1 \in S_1\} = P\{C_1 \in S \text{ or } C_1 \cup \{L\} \in S\} = P\{C_2 \in S \text{ or } C_2 \cup \{\bar{L}\} \in S\} = P\{C_2 \in S_2\}$. The same proof applies to S_2 as well. $\square$

Lemma 2.3. Let $S \in K(n, r)$. Then for some constant c , $c > 0$, cnr bounds the time needed to perform one step of DPP or DPP'. (A *step* of DPP tests if S is empty, if the empty set is in S , if the unit or pure literal rule can be applied, and then applies the appropriate rule.)

Proof. Since each clause contains at most r literals, nr bounds the length of S . In one scan through S the tests of the DPP and DPP' can be performed. The sets S_1 and S_2 can be created in a second pass. $\square$

If the steps of DPP could not be performed in linear time, it would not be possible to infer that the expected time of the DPP is bounded by the expected time of DPP'.

Subsumption is a rule of inference based on the fact that if C_1 and C_2 are in a set of clauses, S , and $C_1 \subseteq C_2$ then S is unsatisfiable iff $S - \{C_2\}$ is. The rule consists of deleting all such "subsumed" clauses C_2 . After a set S is split into sets S_1 and S_2 it might be worthwhile to perform subsumption and possibly delete clauses from S_1 or S_2 . Unfortunately, it is not clear that subsumption can be applied

in linear time. When subsumption is employed, an analysis like the one below will show that the procedure has polynomial expected time performance. However, the additional overhead of subsumption will degrade the bound.

Let $T(n,r)$ be the expected time complexity of the DPP' with the uniform distribution on $K(n,r)$.

Theorem 2.4.

$$T(n,r) = \begin{cases} cnr + \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} T(i,r-1), & n,r > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Proof. By the definition of the expected time complexity of an algorithm,

$$T(n,r) = \sum_{S \in K(n,r)} P\{S\} [\text{the time DPP' takes on } S].$$

Given $S \in K(n,r)$, DPP' first performs one step of the algorithm. This step may cause the algorithm to terminate (in lines 1 or 2 of the algorithm) or it may result in the recursive application of the DPP' to sets S_1 and S_2 . By assuming that the latter occurs, the expected time complexity is being bounded. If the algorithm finds that S_1 is satisfiable it does not have to check the satisfiability of S_2 . We assume the algorithm tests S_1 and S_2 for satisfiability.

The time needed to perform one step of DPP' is cnr (Lemma 2.3). After performing that step the algorithm is

left to test the satisfiability of S_1 and S_2 . Let $S_1 \in K(n_1, r_1)$, $S_2 \in K(n_2, r_2)$, where $1 \leq n_1 < n$, $1 \leq n_2 < n$, and $r_1 = r_2 = r-1$. Since clauses of S are uniformly distributed over $C(r)$, by Lemma 2.2, clauses of S_1 and S_2 are uniformly distributed over $C(r-1)$. Thus, $T(n_1, r-1)$ (respectively, $T(n_2, r-1)$) represents the expected time needed to check the satisfiability of S_1 (respectively, S_2). Therefore,

$$T(n, r) = \sum_{S \in K(n, r)} \text{cnr} + T(n_1, r-1) + T(n_2, r-1) .$$

By collecting terms involving $T(i, r-1)$, $1 \leq i < n$, and noting that

$\sum_{S \in K(n, r)} P\{S\} \text{cnr} = \text{cnr}$, the following recursion is obtained:

$$\begin{aligned} T(n, r) = \text{cnr} + \sum_{i=1}^{n-1} P\left\{S \text{ splits so that } |S_1| = i\right\} T(i, r-1) \\ + \sum_{i=1}^{n-1} P\left\{S \text{ splits so that } |S_2| = i\right\} T(i, r-1) . \end{aligned}$$

However,

$$P\left\{S \text{ splits so that } |S_1| = i\right\} = \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i}$$

since for each clause, with probability $2/3$ (Lemma 2.1), that clause (with $\bar{L}$ deleted) is put into S_1 . Since the same probability applies to S_2 we get

$$T(n, r) = \begin{cases} \text{cnr} + 2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} T(i, r-1) , & n, r > 0 \\ 0 & \text{otherwise.} \end{cases}$$

□

Lemma 2.5. $T(n,r) \leq crT'(n)$, where

$$T'(n) = n + 2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} T'(i) .$$

Proof. The proof is by induction. The base case, $n = 0$, is trivial. Assume the lemma true for $i < n$.

$$T(n,r) = crn + 2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} T(i,r-1)$$

Applying the induction hypothesis,

$$T(n,r) \leq crn + 2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} c(r-1) T'(i)$$

Since all terms are positive,

$$\begin{aligned} &\leq crn + 2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} crT'(i) \\ &= crT'(n) . \end{aligned}$$

□

Our goal is to bound $T(n,r)$. The previous lemma simplified this task by allowing us to consider a function of one variable, $T'(n)$.

Theorem 2.6. $T(n,r) \leq crn^2$.

Proof. It suffices to show that $T'(n) \leq n^2$. This will be done by induction. Direct computation of $T'(n)$ verifies the assertion for $n = 1, \dots, 13$. This will serve as the base case of the induction.

$$T'(n) = n + 2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} T'(i) .$$

By induction,

$$T'(n) \leq n+2 \sum_{i=1}^{n-1} \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} i^2$$

which must be shown to be less than n^2 .

Extending the sum with positive terms,

$$T'(n) \leq n+2 \sum_{i=0}^n \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} i^2 .$$

The sum, $\sum_{i=0}^n \binom{n}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{n-i} i^2$, is the second moment of the binomial distribution with parameter $p = 2/3$. This can be shown equal to [Davis, 1962]

$$p^2 n(n-1) + np$$

(Note that this second moment differs from the variance, $np(1-p)$, which is the second moment centered around the mean.) Substituting, we get

$$\begin{aligned} T'(n) &\leq n+2 \left[\frac{4}{9} n(n-1) + \frac{2}{3} n \right] \\ &= \frac{8}{9} n^2 + \frac{13}{9} n \end{aligned}$$

Observing that because the coefficient of n^2 in this polynomial is less than 1, we have for $n \geq 13$, $(8/9)n^2 + (13/9)n \leq (8/9)n^2 + (1/9)n^2 = n^2$. □

To strengthen our claim that the satisfiability problem is tractable when the DPP is employed, we now show how to obtain polynomial bounds when distributions other than the uniform distribution on the problem instances are assumed. As above we shall still consider $K(n,r)$ as the n -fold

product space of $C(r)$. Furthermore, an analogue of Lemma 2.2 must be valid for the distributions placed on $C(r)$. That is, the removal of an arbitrarily chosen literal from clauses of $C(r)$ distributed according to the pdf on $C(r)$ must result in clauses of $C(r - 1)$ distributed according to its pdf. This condition is a natural one and would only fail if radically different distributions are put on $C(r)$ and $C(r-1)$. Finally, we must assume that $\inf_{i,r} \{p_{ir}, q_{ir}\} = q$ is greater than 0. This condition is equivalent to stating that for any literal L , the probability of its appearance in a clause in $C(r)$ is bounded away from zero as $r \rightarrow \infty$.

Theorem 2.7. Let $p = 1-q$. Under the conditions on the pdf for $C(r)$ above, the expected time of DPP' is bounded by

$$T_p(n,r) = cnr + 2 \sum_{i=1}^{n-1} \binom{n}{i} p^i q^{n-i} T_p(i, r-1)$$

Proof. The proof is identical to that of Theorem 2.4.

In time cnr one step of the DPP' is performed leaving two problems each of size i with probability $\binom{n}{i} p^i q^{n-i}$.

(Actually when removing x_j from clauses in $C(r)$ the probability of obtaining a split set, S_1 , of size i is $\binom{n}{i} \left(1 - p_{jr}\right)^i p_{jr}^{n-i}$. Hence, we are overestimating the size of the split set by using p and q). □

The uniform distribution corresponds to $p = 2/3$.

The expected size of a split set is np . As $p \rightarrow 1$, the result of a split is less effective in terms of reducing the size

of the split sets. This increases the expected time behavior of the algorithm. Let

$$T'_p(n) = n+2 \sum_{i=1}^{n-1} \binom{n}{i} p^i q^{n-i} T'_p(i) .$$

Clearly,

$$T_p(n,r) \leq cr T'_p(n) .$$

Lemma 2.8. Let $f_{p,k}(n) = n+2 \sum_{i=1}^{n-1} \binom{n}{i} p^i q^{n-i} n^k$ where

k is integral and $0 < p < 1$. If $k > -1/\log_2 p$

then $f_{p,k}(n) \leq n^k$, for sufficiently large n .

Proof. $f_{p,k}(n) \leq n+2 \sum_{i=0}^n \binom{n}{i} p^i q^{n-i} n^k .$

The sum $\sum_{i=0}^n \binom{n}{i} p^i q^{n-i} n^k$ is the k^{th} moment

of the binomial distribution. By use of finite difference methods [Riordan, 1958] it can be shown that

$$\sum_{i=0}^n \binom{n}{i} p^i q^{n-i} i^k = \sum_{s=1}^k \left\{ \begin{matrix} k \\ s \end{matrix} \right\} n^{(s)} p^s ,$$

where $\left\{ \begin{matrix} k \\ s \end{matrix} \right\}$ denotes Stirling numbers of the second kind and $n^{(s)} = n(n-1) \dots (n-s+1)$. This sum is a polynomial in n of degree k . The coefficient of the n^k term is $\left\{ \begin{matrix} k \\ k \end{matrix} \right\} p^k$ and since $\left\{ \begin{matrix} k \\ k \end{matrix} \right\} = 1$ we have

$$\begin{aligned} f_{p,k}(n) &\leq n + 2(p^k n^k + \text{lower order terms in } n) \\ &\leq 2p^k n^k + \text{lower order terms in } n . \end{aligned}$$

Hence, for sufficiently large n ,

$$f_{p,k}(n) \leq n^k \Leftrightarrow 2p^k < 1 .$$

$$\Leftrightarrow p^k < 1/2$$

$$\Leftrightarrow k \log_2 p < -1$$

$$\Leftrightarrow k > -\frac{1}{\log_2 p} , \text{ since } \log_2 p < 0 . \quad \square$$

Thus we expect that the following result is obtainable:

$$T_p(n, r) \leq crn^k \quad \text{where} \quad k = \left\lceil -\frac{1}{\log_2 p} \right\rceil$$

Completion of an inductive proof of the statement for a particular p requires establishment of the base case;

Lemma 2.8 then supplies the inductive case. This involves verifying that $T'_p(n) \leq n^k$ for all n less than some N_0 , where N_0 can be computed as in Theorem 2.6.

However, for p such that $\left\lceil -\frac{1}{\log_2 p} \right\rceil + \frac{1}{\log_2 p}$ is small such a verification may not be possible. For these p , we must settle for $T'_p(n)$ bounded by n^{k+1} .

As $p \rightarrow 1$, $\log_2 p \rightarrow 0$ and so k gets large. The growth of k as a function of p is tabulated in Figure 2.1. A table of the function $T'_p(n)$ for $n = 1$ to $n = 50$ and various values of p is Figure 2.2. Figures 2.3, 2.4, 2.5, 2.6 and 2.7 graph $T'_p(n)$ for $p = 0.5, 0.6, 0.7, 0.8$ and 0.9 .

It is helpful to consider a simpler equation that, while not adhering to the definition of the expected time complexity of an algorithm, does capture in a simplified context the essence of the argument made above. Let

$$C_p(n, r) = \begin{cases} cnr + 2C_p(pn, r - 1), & n, r > 0 \\ 0 & \text{otherwise} \end{cases}$$

for $0 < p < 1$. This equation simplifies $T_p(n, r)$ by not summing over all possible sizes of the split sets, but considers only the expected size pn of a split set.

p	$K = \left\lceil -\frac{1}{\log_2 p} \right\rceil$
$0.5 < p \leq 0.71$	2
$0.71 < p \leq 0.79$	3
$0.79 < p \leq 0.84$	4
$0.84 < p \leq 0.87$	5
$0.87 < p \leq 0.89$	6
0.9	7
0.95	14
0.99	69

Figure 2.1

n	T _n (0.1)	T _n (0.2)	T _n (0.3)	T _n (0.4)	T _n (0.5)	T _n (0.6)	T _n (0.7)	T _n (0.8)	T _n (0.9)
0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
2	2.360	2.640	2.840	2.960	3.000	2.960	2.840	2.640	2.360
3	3.613	4.275	4.956	5.569	6.000	6.133	5.883	5.220	4.201
4	4.839	5.849	7.075	8.448	9.750	10.593	10.497	9.138	6.687
5	6.063	7.413	9.186	11.428	13.984	16.248	17.001	14.907	10.039
6	7.291	8.987	11.312	14.472	18.536	22.938	25.629	23.154	14.558
7	8.522	10.574	13.463	17.579	23.323	30.500	36.522	34.620	20.650
8	9.756	12.171	15.641	20.747	28.306	38.809	49.741	50.147	28.864
9	10.993	13.778	17.841	23.973	33.465	47.774	65.297	70.657	39.937
10	12.232	15.392	20.060	27.251	38.782	57.329	83.170	97.137	54.855
11	13.473	17.011	22.292	30.573	44.244	67.428	103.326	130.614	74.928
12	14.715	18.634	24.536	33.933	49.840	78.035	125.730	172.140	101.889
13	15.959	20.260	26.788	37.326	55.556	89.121	150.348	222.776	138.009
14	17.204	21.888	29.048	40.749	61.385	100.662	177.152	283.582	186.246
15	18.450	23.517	31.314	44.197	67.316	112.635	206.117	355.615	250.416
16	19.696	25.147	33.585	47.670	73.344	125.023	237.226	439.922	335.407
17	20.944	26.779	35.861	51.164	79.461	137.807	270.460	537.541	447.424
18	22.191	28.410	38.141	54.679	85.663	150.972	305.808	649.502	594.275
19	23.439	30.042	40.426	58.213	91.945	164.505	343.258	776.828	785.703
20	24.687	31.674	42.714	61.765	98.302	178.393	382.801	920.536	1033.766
21	25.936	33.307	45.006	65.335	104.732	192.624	424.430	1081.642	1353.263
22	27.185	34.939	47.302	68.921	111.230	207.188	468.136	1261.160	1762.211
23	28.433	36.572	49.602	72.522	117.793	222.074	513.914	1460.103	2282.378
24	29.682	38.206	51.905	76.139	124.420	237.273	561.758	1679.489	2939.864
25	30.931	39.839	54.211	79.769	131.107	252.778	611.663	1920.335	3765.747
26	32.179	41.473	56.521	83.413	137.851	268.579	663.623	2183.665	4796.776
27	33.428	43.107	58.835	87.070	144.652	284.670	717.633	2470.503	6076.123
28	34.677	44.742	61.151	90.740	151.505	301.044	773.690	2781.882	7654.196
29	35.925	46.377	63.471	94.421	158.411	317.693	831.788	3118.835	9589.508
30	37.174	48.012	65.793	98.113	165.366	334.613	891.924	3482.404	11949.603
31	38.422	49.648	68.118	101.816	172.369	351.796	954.094	3873.633	14812.042
32	39.670	51.284	70.446	105.530	179.419	369.237	1018.293	4293.571	18265.451
33	40.919	52.921	72.777	109.253	186.514	386.932	1084.518	4743.273	22410.625
34	42.167	54.559	75.110	112.985	193.653	404.875	1152.766	5223.795	27361.699
35	43.415	56.197	77.445	116.727	200.834	423.061	1223.032	5736.203	33247.375
36	44.662	57.835	79.783	120.477	208.056	441.486	1295.315	6281.561	40212.215
37	45.910	59.474	82.123	124.235	215.318	460.146	1369.609	6860.940	48418.000
38	47.158	61.114	84.464	128.002	222.619	479.036	1445.913	7475.417	58045.147
39	48.405	62.754	86.808	131.776	229.958	498.153	1524.223	8126.068	69294.202
40	49.653	64.395	89.153	135.558	237.334	517.492	1604.536	8813.975	82387.393
41	50.900	66.036	91.500	139.347	244.746	537.051	1686.850	9540.225	97570.251
42	52.147	67.677	93.849	143.143	252.193	556.825	1771.160	10305.905	115113.310
43	53.394	69.320	96.199	146.945	259.674	576.812	1857.466	11112.108	135313.865
44	54.642	70.962	98.551	150.755	267.189	597.007	1945.763	11959.927	158497.815
45	55.889	72.605	100.904	154.571	274.737	617.409	2036.050	12850.462	185021.570
46	57.136	74.249	103.259	158.393	282.317	638.014	2128.323	13784.813	215274.039
47	58.383	75.893	105.615	162.221	289.928	658.819	2222.581	14764.083	249678.696
48	59.629	77.537	107.972	166.055	297.570	679.821	2318.821	15789.379	288695.713
49	60.876	79.182	110.330	169.895	305.242	701.019	2417.040	16861.809	332824.187
50	62.123	80.827	112.689	173.740	312.943	722.408	2517.236	17982.485	382604.436

Figure 2.2

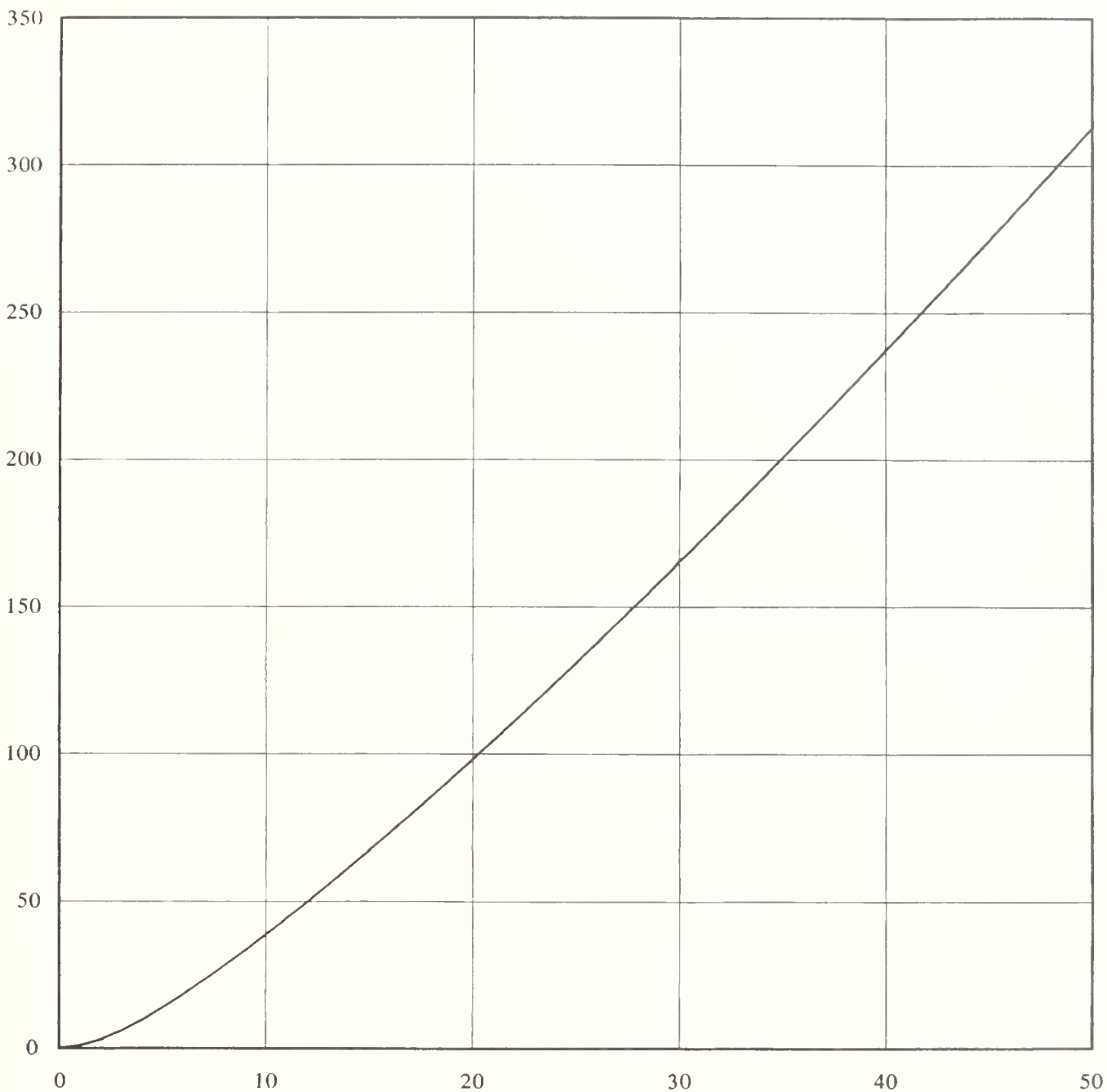


Figure 2.3

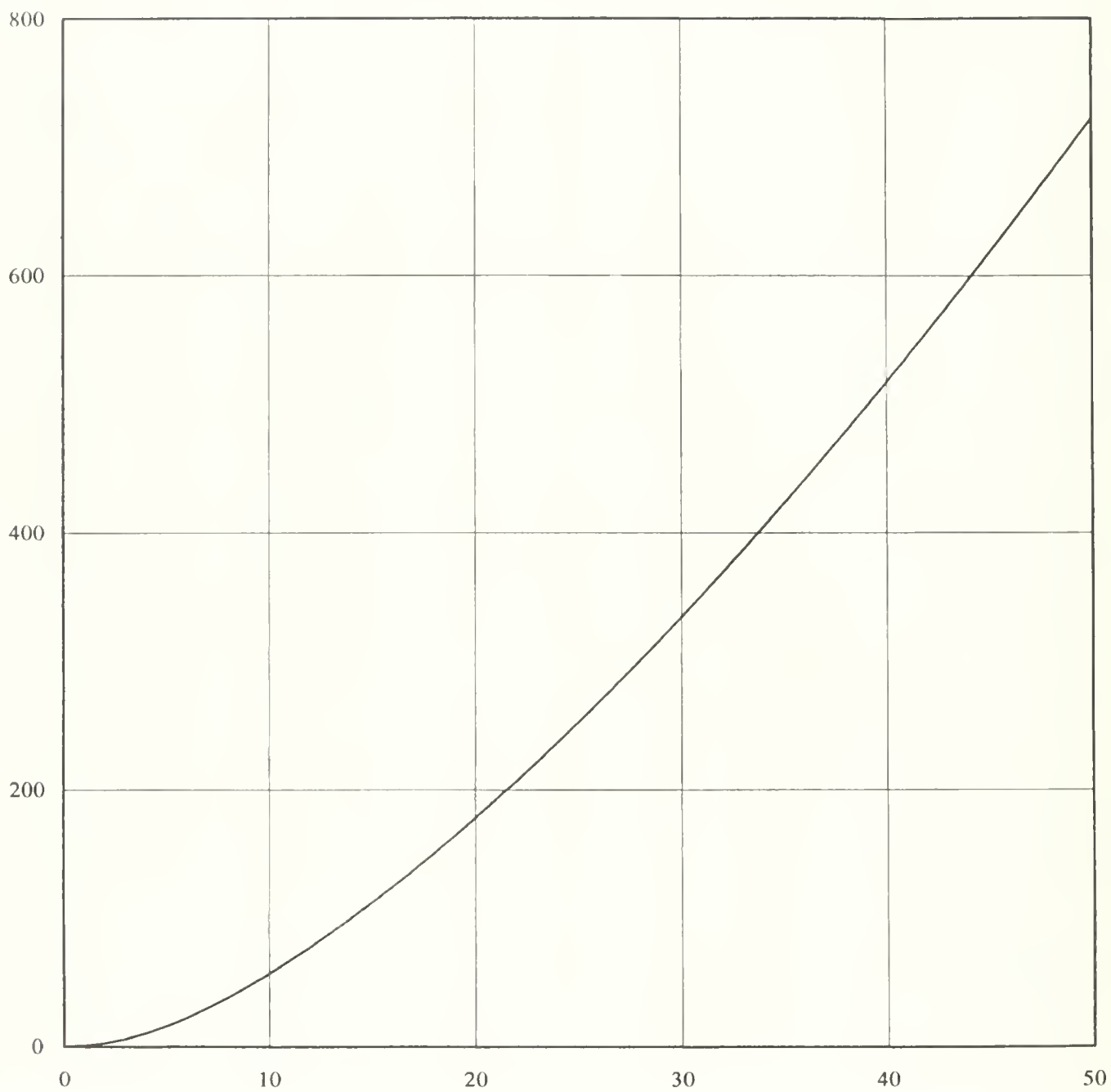


Figure 2.4

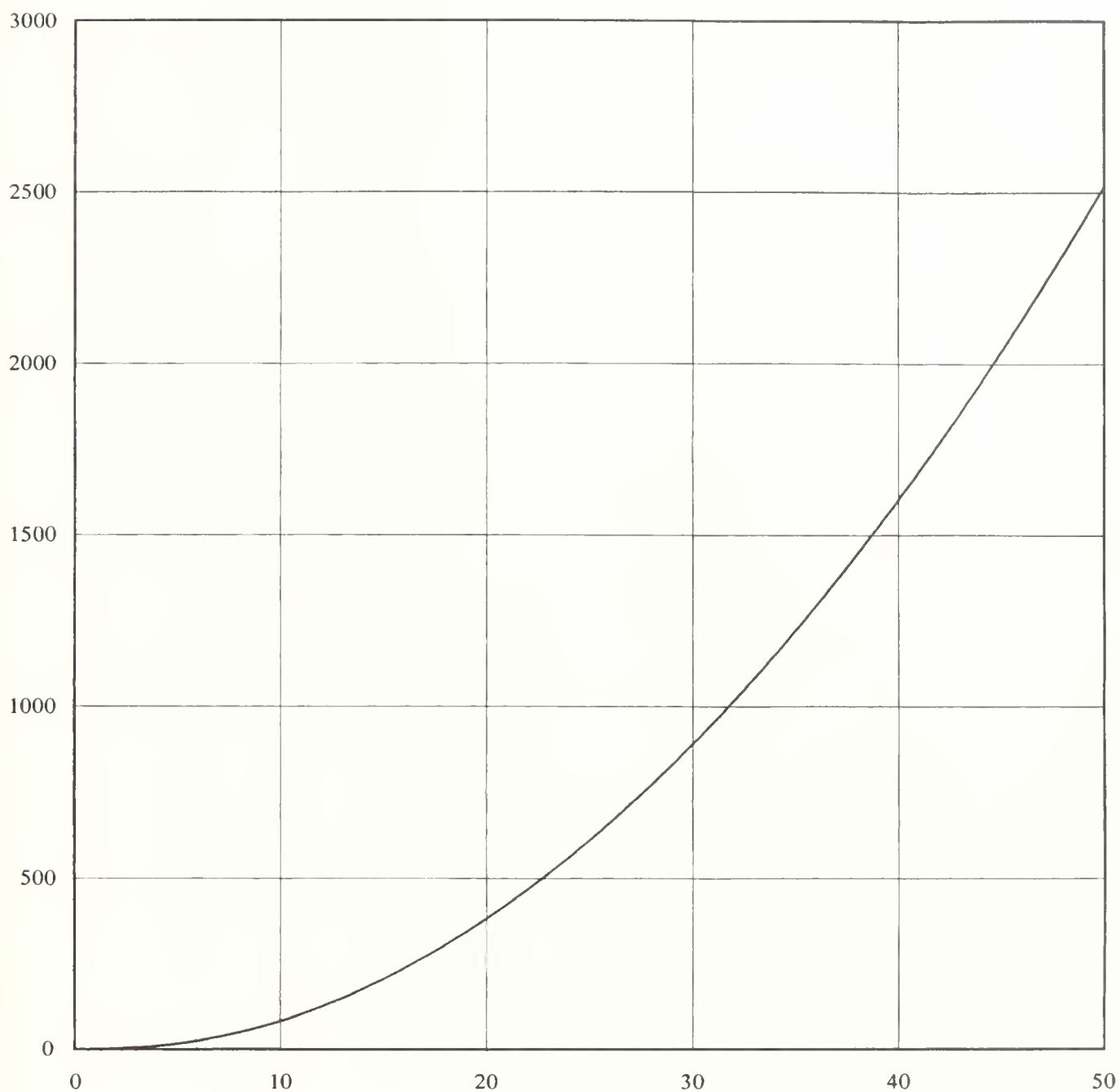


Figure 2.5

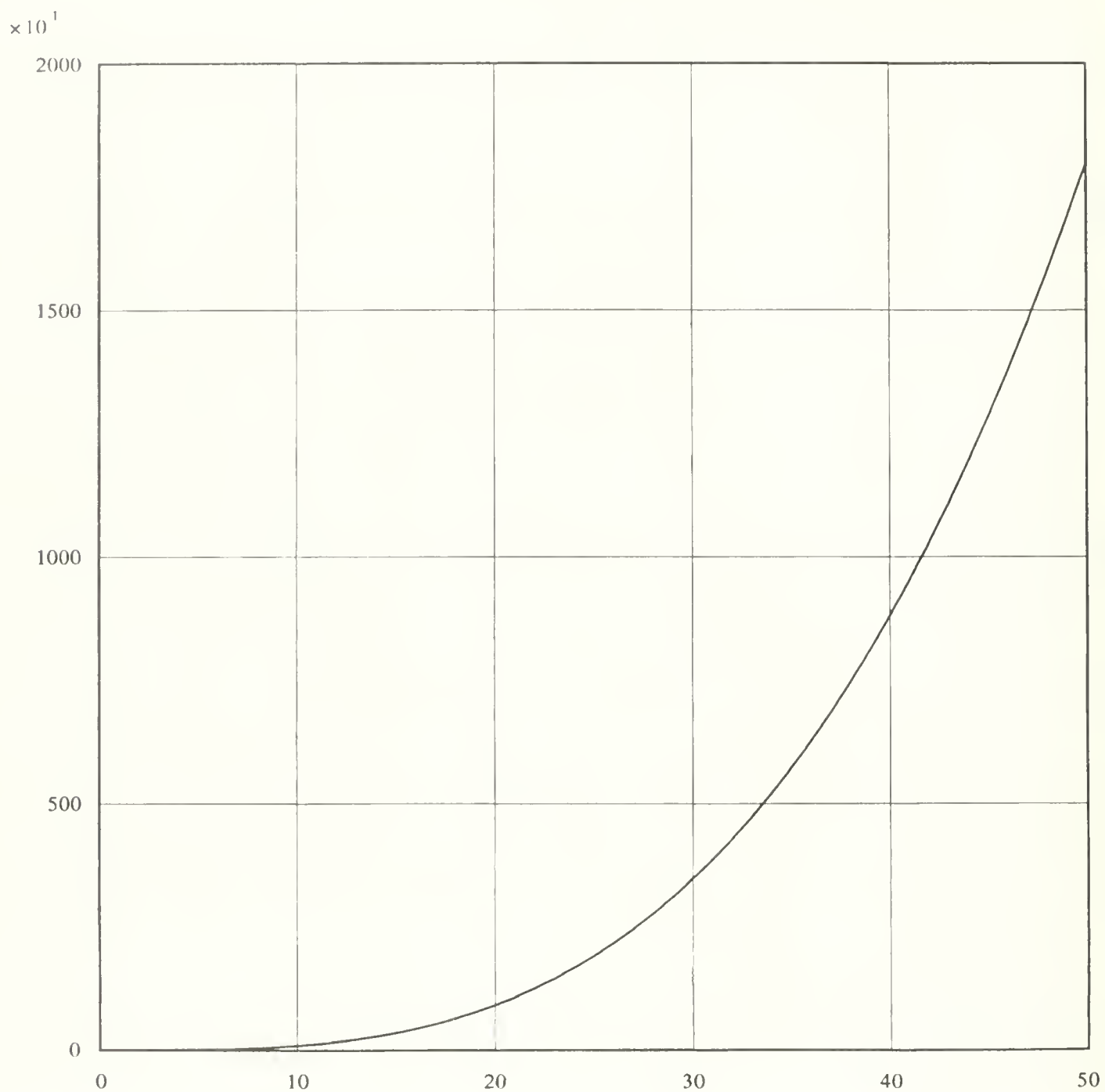


Figure 2.6

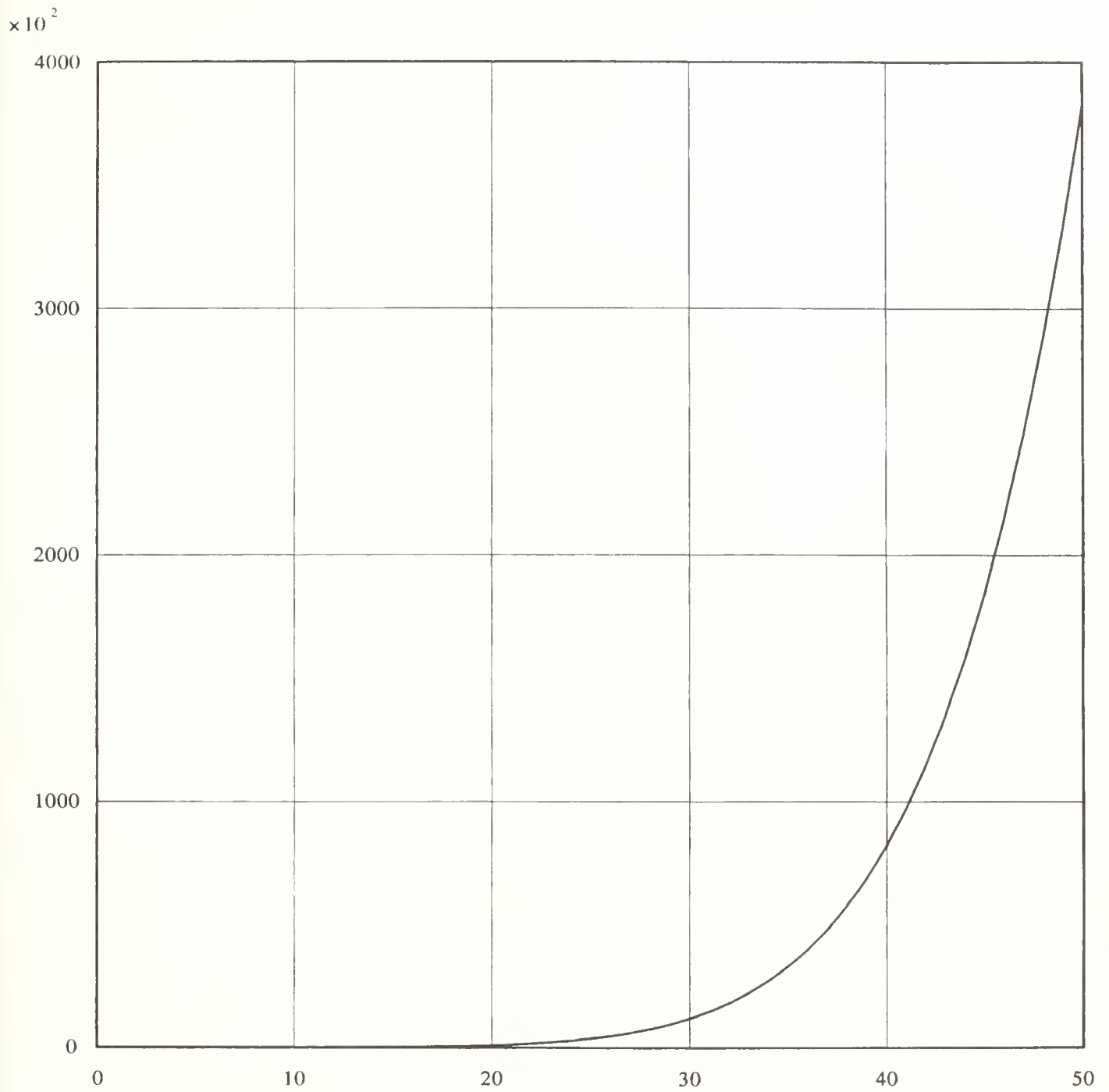


Figure 2.7

Interestingly, it is easy to show that both equations give the same complexity bounds.

Theorem 2.9. $C_p(n, r)$ is $O(rn^k)$ where $k = -\frac{1}{\log_2 p}$

$$\begin{aligned}
 \text{Proof. } C_p(n, r) &= cnr + 2 C_p(pn, r-1) & (n, r > 0) \\
 &= cnr + 2(cpn(r-1) + 2(cp^2n(r-2) + \dots)) \\
 &\leq cnr + 2cpnr + 2^2cp^2nr + 2^3cp^3nr + \dots \\
 &= cnr \sum_{i=0}^m (2p)^i
 \end{aligned}$$

where m satisfies the condition

$$p^m n = 1 \quad \text{i.e.} \quad m = -\log_p n.$$

$$\begin{aligned}
 \sum_{i=0}^m (2p)^i &= \frac{(2p)^{1+m} - 1}{2p - 1} \\
 C_p(n, r) &\leq crn \left[\frac{(2p)^{1-\log_p n} - 1}{2p - 1} \right] \\
 &\leq \frac{2pc}{2p - 1} rn (2p)^{-\log_p n} \\
 &= c' r 2^{-\log_p n}, \quad c' = \frac{2pc}{2p - 1} \\
 &= c' rn^{-\log_p 2} \\
 &= c' rn^{-1/\log_2 p} \quad \text{which is } O(rn^k). \quad \square
 \end{aligned}$$

The variance of the time complexity of an algorithm, AL, measures the deviation of the running time of AL on an arbitrary problem instance from the expected running time of the algorithm. If the variance is small, then instances on which the algorithm's running time is significantly slower (or faster) than the expected running time would be rare. The variance of the DPP has not been computed. But there is evidence that the variance is small. In effect, $C_p(n,r)$ measures the complexity of DPP' under the assumption that the variance is zero. Since $C_p(n,r)$ grows at the same rate as $T_p(n,r)$, it is reasonable to believe that the variance of $T_p(n,r)$ is not large. Second, it is clear from examination of DPP' that the variance of the complexity is due to deviations in the size of the split sets from the expected size. Thus, the variance of the complexity of DPP' is closely related to the variance of the binomial distribution, since the size of a split set is distributed according to this distribution. If X is a binomially distributed random variable, with parameters n and p , then by Chebyshev's inequality, for $\epsilon > 0$,

$$P\left\{|X - np| \geq \epsilon\right\} < \frac{np(1-p)}{\epsilon^2}.$$

So for any fixed k ,

$$P\left\{|X - np| \geq \frac{np}{k}\right\} < \frac{np(1-p)k^2}{n^2 p^2} = \frac{(1-p)k^2}{np}$$

which approaches 0 as $n \rightarrow \infty$. Thus, the probability that a split set deviates from its expected size by more than

a fixed percentage approaches 0 as $n \rightarrow \infty$. This helps explain the relevance of $C_p(n,r)$ to $T_p(n,r)$.

It was hoped that an explicit solution to the recurrence for $T_p(n,r)$ would be derived. Towards this end, the generating function of $T'_p(n)$ was obtained. Starting with the defining relation for $T'_p(n)$,

$$T'_p(n) = \begin{cases} n + 2 \sum_{i=1}^{n-1} \binom{n}{i} p^i q^{n-i} T(i) & , \quad n > 0 \\ 0 & n = 0 \end{cases}$$

we extend the sum to form the complete convolution,

$$(1 + 2p^n) T'_p(n) = n + 2 \sum_{i=0}^n \binom{n}{i} p^i q^{n-i} T'_p(i)$$

$$(1 + 2p^n) \frac{T'_p(n)}{n!} = \frac{1}{(n-1)!} + 2 \sum_{i=0}^n \frac{q^{n-i}}{(n-i)!} \cdot p^i \frac{T'_p(i)}{i!}$$

Let $f(n) = \frac{T'_p(n)}{n!}$ and let $F(z) = \sum_{i=0}^{\infty} f(n) z^n$ be the generating function of $f(n)$. In the notation that follows $Gf(n) = F(z)$. Observe [Jordan, 1947]

$$G \frac{1}{(n-1)!} = z e^z$$

$$G 2 \sum_{i=0}^n \frac{q^{n-i}}{(n-i)!} \cdot p^i f(i) = 2G \sum_{i=0}^n \frac{q^{n-i}}{(n-i)!} p^i f(i)$$

$$= 2G \frac{q^n}{n!} G p^n f(n)$$

$$= 2 e^{qz} F(pz) .$$

$$G(1+2p^n) f(n) = G f(n) + 2 G p^n f(n) = F(z) + 2F(pz) .$$

Since $f(0) = 0$, these equations are valid for $n \geq 0$. Hence,

$$F(z) + 2F(pz) = ze^z + 2e^{qz} F(pz)$$

is a functional identity that must be satisfied by $F(z)$.

Solving for $F(z)$ we get

$$F(z) = ze^z + 2(e^{qz} - 1) F(pz) .$$

Expanding the previous equation by substituting pz for z yields

$$F(z) = ze^z + 2(e^{qz} - 1)(pze^{pz} + 2(e^{qpz} - 1)F(p^2z)) .$$

Performing this substitution n times,

$$F(z) = \left(\prod_{i=0}^{n-1} e^{qp^i z} - 1 \right) 2^n F(p^n z) + \sum_{j=0}^{n-1} (2p)^j ze^{p^j z} \prod_{k=0}^{j-1} (e^{qp^k z} - 1)$$

It can be shown that the first term of this expression does not contribute to the coefficient of z^k , for $k = 1, \dots, n$.

Even with this simplification, it is improbable that an explicit expression for $f(n)$ can be obtained.

We next present a result which shows that in a very strong sense the instances in which the DPP takes exponential time are rare. The next lemma restates the result of Theorem 2.6 in a weakened form, where the size of a problem instance is used to parameterize the sample space.

Lemma 2.10. Let S_n be the set of all sets of clauses of size (i.e. length) n . Let $R(n)$ be the expected time complexity of the DPP with the uniform distribution as the underlying distribution. Then, $R(n) \leq cn^3$.

Proof. Any formula of length n can have at most n distinct atoms and n clauses and so is in $K(n,n)$. Therefore,
 $R(n) \leq T(n,n) \leq cn^3$. □

Let $X(\cdot)$ be a predicate on the set of problem instances, $S = \bigcup_n S_n$ and let q_n be the probability that $X(I)$ does *not* hold when I is drawn from S_n . Following Karp [Karp, 1976] a predicate X holds *almost everywhere* if $\sum_{n=1}^{\infty} q_n$ converges. This condition implies that if we draw an infinite sequence of problem instances, one of each size, then with probability 1, the predicate X will fail on only finitely many.

Lemma 2.11. Let $f(n)$ be the expected time complexity of algorithm AL on S_n , problem instances of size n . Assume that $f(n) \leq cn^k$. For $\alpha > 0$, let $X_\alpha(I)$ be the predicate "AL computing on $I \in S_n$ takes time bounded by $e^{\alpha n}$." Then $X_\alpha(I)$ holds almost everywhere.

Proof.
$$f(n) = \sum_{I \in S_n, X(I) \text{ true}} P\{I\}[\text{running time of AL on } I] + \sum_{I \in S_n, X(I) \text{ false}} P\{I\}[\text{running time of AL on } I] \leq cn^k.$$

Dropping the first term and observing that

$$\sum_{I \in S_n, X(I) \text{ false}} P\{I\}[\text{running time of AL on } I] \geq q_n e^{\alpha n},$$

$X(I) \text{ false}$

the following inequality is obtained:

$$0 \leq q_n e^{\alpha n} \leq cn^k \quad \text{or} \quad 0 \leq q_n \leq \frac{cn^k}{e^{\alpha n}}.$$

So if $\sum_{n=0}^{\infty} \frac{cn^k}{e^{\alpha n}}$ converges, then $\sum_{n=0}^{\infty} q_n$ will converge.

But by applying the ratio test,

$$\lim_{n \rightarrow \infty} \left| \frac{\frac{c(n+1)^k}{e^{\alpha(n+1)}}}{\frac{cn^k}{e^{\alpha n}}} \right| = \lim_{n \rightarrow \infty} \frac{(n+1)^k}{e^{\alpha} n^k} = \frac{1}{e^{\alpha}} \lim_{n \rightarrow \infty} \frac{(n+1)^k}{n^k} = \frac{1}{e^{\alpha}} < 1. \quad \square$$

Corollary 2.12. For any $\alpha > 0$, the DPP takes time less than $e^{\alpha n}$ almost everywhere on a set of clauses of length n .

Proof. Apply the two previous lemmas. $\square$

Performing an analysis of an algorithm often yields deeper insights into the nature of the algorithm. The analysis of the DPP reveals that the worst case behavior occurs for sets of clauses on which the splitting rule does not effectively reduce the size of the set. This in itself suggests a strategy for choosing the literal on which splitting is to occur, namely, choose the literal L that occurs most frequently in the set. This rule can be implemented economically. For each literal pair, L and $\bar{L}$, counts of their occurrences in the set are easily maintained. Other factors can be incorporated into this strategy. For example, splits that allow the pure literal or unit rule to be applied subsequently may be favored. The analysis also warns that the literal selection procedure should not

be too complicated; the selection should be made in linear time. The usefulness of subsumption is also called into question, since one step of the DPP may not take linear time if subsumption is employed.

Reducibility (see Ch. I) is an important notion employed in the study of the worst case behavior of an algorithm. If problem A can be polynomially reduced to problem B and a polynomial (worst case) exists for B, then one exists for A as well. Unfortunately, such arguments cannot be made in the average case. Instances of problem A are typically mapped by the reduction into a subclass of the instances of problem B, violating the distributional assumptions underlying the analysis. Reductions frequently map problem instances into sets of clauses in such a way that as the size of the problem instances increase, the probability that a literal appears in a clause in the corresponding set goes to zero. This violates one of the conditions required of the distribution in order for the average case analysis to be applicable.

This of course does not in any way imply that other NP-complete problems may not have algorithms which have polynomial average case behavior. We believe that at least some of them do. However, it is the "divide and conquer" nature of the DPP that enables us to perform this analysis. Algorithms of this type should be sought for other NP-complete problems.

III. PROOF SYSTEMS AND THE DAVIS-PUTNAM PROCEDURE

In this chapter, the relationship of the Davis-Putnam procedure, viewed as a proof procedure, to other proof systems for the propositional calculus is examined. Reckow, in his thesis [Reckow, 1976], assesses the relative power of many classical proof systems. If proof system A p-simulates B then A is at least as powerful as B. His investigation shows that Frege systems are the most powerful since they can simulate all other systems. Gentzen, natural deduction, sequent, full resolution and extended regular resolution are of roughly comparable power. Exponential lower bounds on proof length have not been established for these systems. Recall, that if $NP \neq \overline{NP}$ then all proof systems are not polynomially bounded. Analytic tableaux and regular resolution have known exponential worst cases, but are still superior to full truth tables. The Davis-Putnam procedure has also been classified by Reckow in the partial ordering of proof systems determined by the p-simulates relation. However, the Davis-Putnam procedure considered in his study does not correspond to what has been called the DPP in this thesis. The difference can be traced to historical reasons. In 1960, the Davis-Putnam procedure was introduced in a paper called "A Computing Procedure for Quantification Theory" [Davis and Putnam, 1960].

The unit clause rule and the pure literal rule were introduced. However, in place of the splitting rule there is a rule called "rule for eliminating atomic formulas":

Write a formula, ϕ , given in CNF as $(A \vee L) \& (B \vee \bar{L}) \& R$, for some literal L . That is, choose L , and collect all clauses containing L . Then factor L from those clauses leaving a conjunction A . B is the conjunction of clauses obtained by factoring $\bar{L}$ from all clauses containing $\bar{L}$. R is the conjunction of clauses not containing L or its complement. Then ϕ is satisfiable iff $(A \vee B) \& R$ is satisfiable. The formula $(A \vee B) \& R$ is then restored to CNF and a new rule is applied. The rule for eliminating atomic formulas is sound and complete; the procedure terminates when either the empty clause is derived or the set becomes empty. The difficulty with this rule is that restoration of $(A \vee B) \& C$ into CNF requires distributing A through B which frequently results in a combinatorial explosion of new clauses added to the set. Many of these clauses are tautologies or subject to subsumption. The connection of this rule with resolution is evident; the set of clauses obtained by putting $(A \vee B) \& R$ into CNF is precisely the set obtained by forming all resolvents of clauses of ϕ in which L is the annihilated literal, and then dropping all clauses containing L or its complement. Thus, this version of the Davis-Putnam procedure can be described as a resolution procedure which we call DPP":

```

procedure DPP"(S);
  repeat forever;
1.  if S =  $\emptyset$  then return satisfiable end if;
2.  if  $\emptyset \in S$  then return unsatisfiable end if;
3.  Choose a literal L appearing in S.
    T := {C | C  $\in$  S & L  $\in$  C};
    R := {C1 - {L}  $\cup$  C2 - { $\bar{L}$ } | C1  $\in$  T & L  $\in$  C1 & C2  $\in$  T &  $\bar{L} \in$  C2};
    S := S - T  $\cup$  R;
4.  Remove all subsumed and tautological clauses;
    end repeat;
  end DPP";

```

Step 4 is optional. This algorithm can be adopted to include the unit and pure literal rules. The algorithm as presented has been the basis of all the complexity work done on the Davis-Putnam procedure [Cook, 1971; Cook and Reckow, 1974; Galil, 1977; Simon, 1971]. Fortunately, the results reported by the above researchers do not depend upon the exclusion of the two rules.

In 1962, Davis, Logemann and Loveland [Davis, Logemann and Loveland, 1962] found the rule for eliminating atomic formulas increased the number and lengths of the clauses too quickly in their theorem prover. Thus, they replaced it with the splitting rule. In addition to keeping the number of new clauses generated manageable, it also increased the frequency of application of the unit clause rule.

One of the few similarities between DPP and DPP" is that they both eliminate every occurrence of an atom in each application of a rule. As a result they are both regular resolution procedures. It has just been shown how to regard the 1960 procedure as a resolution procedure; we now do the same for DPP.

We will first show how to modify the DPP procedure to generate resolution refutations with only a constant factor increase in the running time of the procedure. The resulting resolution refutation will be regular. A closer examination of the proof will reveal what is the proper resolution analogue to DPP. The recursive structure of the procedure will be used to recursively generate the resolution refutation. Suppose the DPP, given an unsatisfiable set S , splits S on literal L into sets S_1 and S_2 . Since S_1 and S_2 are each unsatisfiable, the DPP can inductively generate resolution refutations for these sets. Restoring the literal $\bar{L}$ (L) into the appropriate clauses in the derivation, refutations from S_1 (S_2) yield resolution derivations of $\bar{L}$ and L from S . Resolving L and $\bar{L}$ derives the empty clause and so completes the construction of the refutation from S . The complete algorithm is described as the procedure $\text{RES}(S, \text{deriv})$.

```

procedure RES(S,deriv);

1.  if S =  $\emptyset$  then deriv := ( );
    return satisfiable; end if;

2.  if  $\emptyset \in S$  then deriv := ( $\emptyset$ );
    return unsatisfiable; end if;

3.  if the pure literal rule applies to S then
    if RES( $S_1$ ,deriv) = satisfiable
        then return satisfiable;
        else return unsatisfiable;
    end if;
end if;

4.  if the unit rule applies to S then
    if RES( $S_1$ ,deriv1) = satisfiable
        then return satisfiable;
        else deriv := (restore  $\bar{L}$  to clauses of deriv1
            (yielding a derivation of  $\bar{L}$ ) append a
            resolution with L);
        return unsatisfiable;
    end if;

5.  else (the splitting rule applies) if
    RES( $S_1$ ,deriv1) = satisfiable or
    RES( $S_2$ ,deriv2) = satisfiable then
    deriv := ( ); return satisfiable;
    else deriv = (restore  $\bar{L}$  to clauses of
    of deriv1 , restore L to clauses of deriv2 ,
    append them together, resolve L with  $\bar{L}$ );
    return unsatisfiable; end if;
end if;
end RES;

```

Theorem 3.1. There is an algorithm that generates resolution refutations of an unsatisfiable set of clauses, S , whose running time is within a constant factor of the running time of the DPP.

Proof. The above procedure may not achieve the bound specified by the theorem because of the necessity of scanning the derivation repeatedly to restore deleted literals. This problem is easily overcome by initially inserting the complete clause, without any of its literals deleted, into the derivation. Hence a derivation is built from constituent derivations in step 5 by appending the two together and adding an additional resolution between the derived clauses. The refutation constructed in this way will be identical to the one generated by RES.

□

In this chapter our concern is with proof length; that is our measure of strength of proof systems. We have not, as yet, described what a DPP proof is. The definition of proof requires that there be an algorithm which verifies the validity of a purported proof in polynomial time. We shall consider a *DPP proof* to be a tree with nodes labeled with the sets derived by application of the rules in the execution of the procedure. The given set is placed at the root. It has either one or two descendants, depending upon which rules have been applied. The descendant sets S_1 and S_2 are modified so that literals deleted from

clauses are only marked as such. This facilitates the generation of the resolution refutation. Alternatively, a DPP proof could be given as a tree in which each node contains the literal eliminated at the subsequent step of the procedure together with the original set of clauses. This puts the burden of reconstructing the sets on the proof checker. This can be done in polynomial time in the size of the tree and the size of the original set. It is easily checked that both of these descriptions are reasonable notions of proof, consistent with the definition of proof system. Given a DPP proof as originally described (i.e. with sets at each node), an endorder traverse on the tree can turn it into a regular tree resolution proof. This is done by the following procedure:

```

procedure TREE-PROOF(DPP-TREE);

1.  Let S be the set of clauses at the root of DPP-TREE;
2.  if DPP-TREE has a single node then
        restore the deleted literals to the empty clause in S;
        label the node with this clause;
        return; end if;
3.  if the root node of DPP-TREE has two descendants,
     $N_1$  and  $N_2$  , then
        TREE-PROOF( $N_1$ );           /* endorder traverse */
        TREE-PROOF( $N_2$ );
        label the root node with the resolvent of the
        clauses labeling its descendants;
        return; end if;

```

```

4.  if the root node has one descendant,  $N_1$  then
      if the descendant was obtained by the unit rule then
          TREE-PROOF( $N_1$ );
          Add the unit clause of S with its deleted
          literals restored as a new descendant of the
          root;
          label the root with the resolvent of the 2
          clauses labeling the descendants;
          return;  end if;
      else /* the descendant was obtained by the pure
          literal rule */
          TREE-PROOF( $N_1$ );
          delete the root node. Its ancestor will link up
          with its single descendant.
          return;  end if;
      end TREE-PROOF;

```

This procedure is performing an endorder (left descendant, right descendant, root) traversal of a DPP tree replacing the sets labeling the nodes with the clauses of a regular tree refutation. At leaf nodes, a clause from the original set replaces the set of clauses at that node. The clause chosen is the one which, in the execution of the DPP, has had all its literals dropped. Once the leaf nodes are specified, the clauses labeling nonleaf nodes are computed as the resolvents of its descendants. Application of the

pure literal rule does not add a resolution to the proof since its effect is to locate an unsatisfiable subset of the original set. The proof of the correctness of this procedure is based on the proof of the previous theorem. Figure 3.1 is an example of a DPP refutation and the corresponding regular tree refutation.

Theorem 3.2. Regular tree resolution p-simulates the DPP.

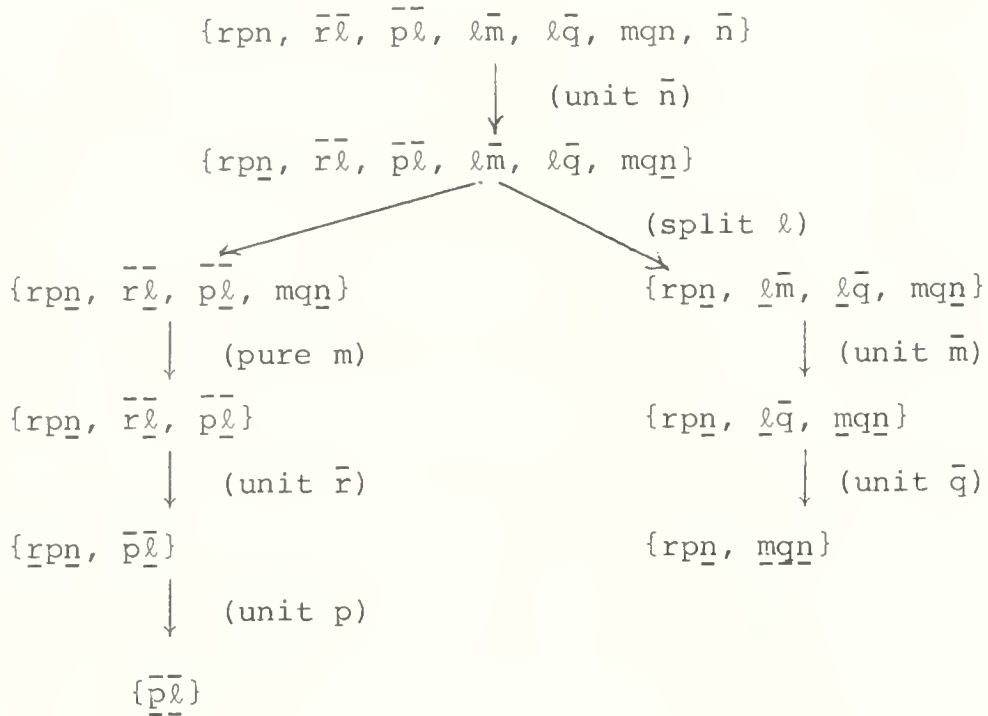
Proof. The procedure TREE-PROOF does the simulation. By construction, the resulting resolution refutation is a tree. Because the DPP completely removes the occurrence of an atom in the derived sets, the resolution refutation is regular. □

It has been remarked in Chapter I that a difficulty with tree resolution is the repetition of the derivation of any clauses needed more than once in a derivation. DPP has this problem as well since a split may be performed yielding two sets and the same clause may have to be derived in both of these sets.

Theorem 3.3. DPP p-simulates regular tree resolution.

Proof. Consider the alternate description of a DPP proof as a tree whose nodes are labeled with the literal annihilated at the subsequent step of the algorithm. Since we can construct a standard DPP proof from this version in polynomial

DPP REFUTATION



TREE RESOLUTION REFUTATION

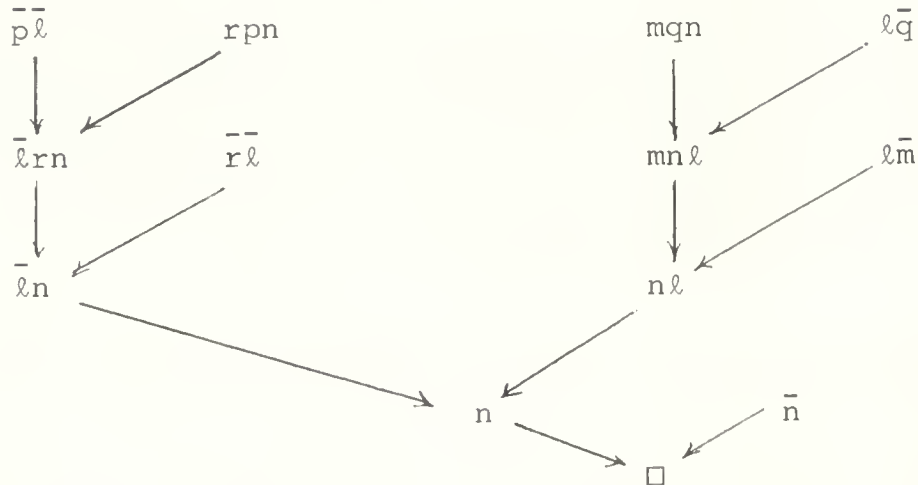


Figure 3.1. Underlined literals have been marked deleted in DPP refutation.
 (Example taken from [Shostack, 1976].)

time, it is only necessary to show how a regular tree resolution proof can be p-simulated by this proof scheme. This is accomplished by replacing the resolvent clause at a nonleaf node by the literal annihilated in forming that resolvent and deleting all the leaf nodes. It is easily proved by induction that the resulting tree does correspond to a correct DPP refutation. □

Refutation graphs were defined by Shostack [Shostack, 1976] as a convenient formalism from which to study theorem proving methods. It is easy to show how a refutation graph can be constructed from a Davis-Putnam refutation in linear time. Shostack does this implicitly in his construction of a refutation graph for unsatisfiable sets [Shostack, 1976]. Thus, refutation graphs p-simulate the DPP. Given a refutation graph, other types of resolution refutations can be generated in polynomial time. Most notable among these is t-linear resolution [Kowalski and Kuehner, 1971]. These simulation results are interesting because the DPP is an effective algorithm as well as a proof system. These simulation results imply that efficient algorithms exist for these other proof systems as well. Of course there is little value in running the DPP and then simulating the results in some other proof system. These results suggest that efficient algorithms that are based directly on these other proof systems may be inferred from the DPP.

The DPP cannot p-simulate refutation graphs. In [Kowalski and Kuehner, 1971] it is shown that t-linear

resolution is as powerful as regular resolution. Corresponding to any t-linear proof there is a refutation graph [Shostack, 1976]. Thus, refutation graphs p-simulate regular resolution. Tseitin [Tseitin, 1968] has shown regular tree resolution, and hence, the DPP cannot p-simulate regular resolution.

From the point of view of power of proof systems, regular tree resolution and the DPP are identical. The proofs of Theorems 3.2 and 3.3 demonstrate that a structural isomorphism exists between the two systems.

Both versions of the Davis-Putnam procedure, DPP and DPP", are instances of regular resolution procedures. The DPP" was rejected on practical grounds in favor of DPP. It will be shown that with respect to the p-simulate relation the two systems are incomparable. (These results will be established for DPP" without subsumption.)

Cook has shown that DPP" without subsumption must generate exponentially long proofs on the following class of problem instances [Cook, 1971]. Let S_m^ℓ be the set of purely positive or purely negative clauses of length ℓ over an atom set containing m atoms, $p_1, \dots, p_m$. That is,

$$S_m^\ell = \{p_{i_1} p_{i_2} \dots p_{i_\ell} \mid 1 \leq i_1 < i_2 < \dots < i_\ell \leq m\} \cup \{\bar{p}_{i_1} \bar{p}_{i_2} \dots \bar{p}_{i_\ell} \mid 1 \leq i_1 < i_2 < \dots < i_\ell \leq m\}$$

Note that S_m^ℓ is unsatisfiable iff $m \geq 2\ell - 1$. If $m < 2\ell - 1$,

then assigning the value True to exactly $\lfloor m/2 \rfloor$ atoms assures that every clause must contain an atom made True by the assignment as well as one made False. Thus all clauses are satisfied.

Conversely, if $m \geq 2\ell - 1$ then for any assignment there is a clause left unsatisfied. If half or more of the atoms are assigned True (False) then one of the purely negative (positive) clauses will be False.

Cook has shown that after the elimination by the DPP" (without subsumption), of any sequence of $\ell - 1$ literals, the number of clauses in the resulting set will grow faster than any polynomial in the size of the original set.

Next we examine the performance of the DPP on this example. For simplicity, assume that m is even and that $\ell = \frac{1}{2} m$. S_m^ℓ is then unsatisfiable. The crudest estimate of the length of the DPP proof assumes that 2^m splits are made and that the size of a split set is $2 \binom{m}{\ell}$, the size of the original set of formulas. Thus, the length of the DPP proof is bounded by $2^{m+1} \binom{m}{\ell}$ or $2^{2\ell+1} \binom{2\ell}{\ell}$. But by Stirling's formula,

$$\begin{aligned} \binom{2\ell}{\ell} &= \frac{(2\ell)!}{\ell! \ell!} \approx \frac{\sqrt{4\pi\ell} \left(\frac{2\ell}{e}\right)^{2\ell}}{2\pi\ell \left(\frac{\ell}{e}\right)^{2\ell}} \\ &= \frac{4^\ell}{\sqrt{\pi\ell}} \end{aligned}$$

Let $n = 4^\ell / \sqrt{\pi\ell}$; then

$$2^{m+1} \binom{m}{\ell} \approx 4^{2\ell} \cdot \frac{4^\ell}{\sqrt{\pi\ell}} = \frac{4^{3\ell}}{\sqrt{\pi\ell}} \leq n^4.$$

So the length of a DPP proof is bounded by a polynomial in the length of the instance.

Theorem 3.4. DPP" cannot p-simulate DPP.

Proof. Any proof of $S_{2\ell}^\ell$ by the DPP" is not within a polynomial of the length of the DPP proof. □

DPP" generates $O(2^{2^\ell})$ clauses for $S_{2\ell}^\ell$, most of which are subsumed by others generated. The DPP" with subsumption will generate a short proof for $S_{2\ell}^\ell$.

Theorem 3.5 (Tseitin, Reckow). Regular tree resolution cannot p-simulate DPP".

Corollary 3.6. DPP cannot p-simulate DPP".

Tseitin [Tseitin, 1968] has found a class of instances $\{S_i\}$ for which there exists a $c > 0$ such that if n is the length of the shortest regular resolution proof of S_i , then the length of every regular tree resolution proof is greater than $2^{c(\log n)^2}$. Reckow has remarked that DPP" without subsumption realizes the shortest regular resolution proof of S_i . □

As these results show, DPP and DPP" are substantially different proof procedures. As regular resolution procedures they both are subject to Tseitin's result, as improved by Galil [Galil, 1977; Tseitin, 1968]; namely, there is a class of instances on which any regular resolution proof

must contain 2^{cn} clauses, for some $c > 0$, and infinitely many n , where n is the size of the instance.

In Chapter II it was remarked that application of the splitting rule with a literal that occurs frequently in a set of clauses is intuitively desirable because this will tend to minimize the number of clauses that fall into both split sets. On the other hand, such a literal will be a poor selection of a literal to annihilate for DPP" because a new clause is added to the set for every pair of complementary literals. For DPP" the opposite heuristic is applicable; eliminate the literal that occurs least frequently. This will tend to minimize the growth of the set. These heuristic considerations and the above results suggest that together these two procedures will work very well. When a literal that is a good candidate for application of the splitting rule is present, apply that rule. When a literal appears infrequently, it should be eliminated by the rule for elimination of atomic formulas (DPP").

The question of the power of a proof system is important theoretically because of the connection to the $NP = \overline{NP}$ question. However, since regular resolution has been proven exponential, this question cannot be answered by investigating the DPP. The results of Chapter II become interesting in light of the weakness of the DPP as a proof system. An interesting open question is: Is there a more powerful proof system which yields an algorithm superior to the DPP, or does the greater nondeterminism of these systems preclude this possibility?

IV. HORN SETS

4.1. The DPP and Horn Sets

Many mathematical statements are of the form $(A_1 \& A_2 \& \dots \& A_n) \rightarrow B$, where $A_1, A_2, \dots, A_n, B$ are atoms, i.e. given hypotheses $A_1, A_2, \dots, A_n$ the conclusion B follows. When converted to CNF, statements of this form become a single conjunct, $\bar{A}_1 \bar{A}_2 \dots \bar{A}_n B$. A clause with at most one positive literal is a *Horn clause*. A set containing exclusively Horn clauses is a *Horn set*. A theorem prover dealing with ordinary mathematics will encounter sets made up predominantly of Horn clauses, and, in fact, the sets encountered will often be Horn sets. As a result, Horn sets have undergone special study. A *unit (input) resolution derivation* is a resolution derivation in which one parent clause of every resolvent is a unit (input) clause. In [Henschen and Wos, 1974] it is shown that if S is an unsatisfiable Horn set, then S has both unit and input resolution refutations. It follows that any unsatisfiable set must contain a unit clause. Thus the DPP, given an unsatisfiable Horn set will on the first step be able to apply the unit rule. The set S_1 derived from application of this rule will also be an unsatisfiable Horn set, since every clause in S_1 is a subset of a clause in S . S_1 must also contain a unit clause. It follows that

at every step of the DPP, the unit rule will be applicable. If at any step the unit rule does not apply to a Horn set, the set must be satisfiable. Construction of an efficient algorithm to test the satisfiability of Horn sets can be based on the DPP utilizing these observations. The algorithm simply continues to apply the unit rule until the set is shown to be unsatisfiable or the unit rule cannot be applied. There is no necessity to consider other rules. The choice of data structures to represent S can take advantage of this situation. Clauses are represented as pointer structures. Each clause contains a field holding its current length. All the literals occurring in S associated with the same atom are linked together. The presence of a literal in a clause is indicated by a pointer to the next clause containing that literal (or its complement). Using this data structure, application of the unit rule is very efficient. Starting at the unit clause, the links from the unit literal are followed to its occurrences in the other clauses either marking the literal deleted from these clauses and decreasing the length of the clause by one, or dropping the clause entirely (by making its length zero). As the links are followed all new unit clauses are noted so they can be used to initiate subsequent steps. With this representation every literal is examined at most once. This establishes

Theorem 4.1. Horn sets can be tested for satisfiability in linear time.

It will nevertheless be efficient to apply the DPP without any special modifications for Horn sets, but the best upper bound obtainable for the worst case performance will be cubic in the size of the set. The complexity goes up one degree because of the lack of any special data structuring which allows for unit rule application in sublinear time. Another degree will be lost because a set will not be found satisfiable if the unit rule does not apply. The splitting rule will be applied and the algorithm will proceed normally, finding a satisfying assignment.

Lemma 4.2. The DPP will run in $O(\ell^2)$ time on unsatisfiable Horn sets of size ℓ .

Proof. At each step the unit rule will be applied. This takes time linear in the length of the set. Since the unit rule leaves only one set to test, the rule will be applied r times, where r is the number of distinct atoms in the set. Since $r \leq \ell$ the algorithm runs in $O(\ell^2)$ time. $\square$

Theorem 4.3. The DPP, without modification, will take at most cubic time, in the worst case, to test the satisfiability of Horn sets.

Proof. The DPP performs a depth first search of possible satisfying valuations. If it ever finds such a valuation it halts. The DPP realizes the worst case for satisfiable

Horn sets when only the last examined valuation satisfies the set. For this to occur, whenever the splitting rule is applied the first set tested (say S_1) must be unsatisfiable and the second S_2 must be satisfiable. Let ℓ be the length of a given set S . Let $G(\ell)$ be the worst case time complexity of the DPP on Horn sets. S_1 is unsatisfiable, Horn and has a maximum length of $\ell-3$. Therefore, by Lemma 4.2, S_1 is found unsatisfiable in time $k(\ell-3)^2$ for some constant k . Thus we get the following recurrence for $G(\ell)$:

$$G(\ell) = \begin{cases} c\ell + k(\ell - 3)^2 + G(\ell - 1), & \ell > 2 \\ 0 & \text{otherwise} \end{cases}$$

$$G(\ell) \leq \sum_{i=1}^{\ell} ci + \sum_{i=1}^{\ell-3} k i^2, \text{ which is } O(\ell^3). \quad \square$$

Whether this upper bound is achieved remains an open problem. It is reasonable to expect very good (quadratic) performance by the DPP on Horn sets.

4.2. Proof Systems for Horn Sets

It is well known [Chang and Lee, 1973] that a set of clauses has a unit refutation iff it has an input refutation. Horn sets have unit and input refutations. A converse can be stated with the aid of a few definitions. Given a set S of clauses and a set A of atoms occurring in S , a *renaming* of S is the set of clauses resulting from the replacement in S of every literal whose atom appears in A by its negation. Renaming preserves satisfiability. A set of clauses is *minimally unsatisfiable* if it is unsatisfiable, but any proper subset of it is not. If S is minimally unsatisfiable and has a unit refutation then S is the renaming of a Horn set [Henshen and Wos, 1974]. The minimality condition is natural since an unsatisfiable Horn set can be trivially augmented by a non-Horn clause and still have a unit refutation. We can conclude that S is a Horn set if S has a unit refutation where the unit clauses are positive literals. From the viewpoint of studying proof systems, an investigation of unit and input refutations can be made by examining their behavior exclusively on Horn sets. As proof systems, input and unit resolution are incomplete. Although they are both powerful systems over the domain for which they are complete (i.e. sets renamable as Horn sets), we shall see that input resolution proofs are generally shorter than unit proofs.

Input resolution corresponds closely to the generation of resolution refutations using the DPP; unit resolution corresponds to the use of the DPP".

Theorem 4.4. Let S be an unsatisfiable Horn set of length ℓ (ℓ is the total number of literals in S). S has a unit resolution refutation in which there are at most ℓ applications of the resolution rule.

Proof. Consider the DPP" refutation of S in which at each step the atom corresponding to the unit clause is annihilated. Assuming that all subsumed clauses have been removed, the unit clause containing the literal L will be the only occurrence of L in the set. Thus, a step of DPP" will resolve L with every clause containing $\bar{L}$ and then drop all the parents of the resolutions. If k of the clauses contain $\bar{L}$, k resolutions will be performed; the resulting set, S_1 , will have length $\ell - k - 1$ (k literals and the unit clause are removed). The same procedure is repeated at the next step. It is easily seen that at most ℓ resolutions are possible. More precisely, if r is the number of distinct atoms in S , and hence the number of steps performed, then the maximum number of resolutions is $\ell - r$. □

The sharper bound, $\ell - r$, is attained by the set $S = \{p_1, p_1\bar{p}_2, \dots, p_{r-1}\bar{p}_r, p_r\}$.

Theorem 4.5. Let S be an unsatisfiable Horn set with r distinct literals. Then there is an input resolution refutation with at most r applications of the resolution rule.

Proof. Consider the resolution refutation obtainable from the DPP procedure. We have previously observed that for an unsatisfiable Horn set, DPP will apply the unit rule at every step. There will be one resolution for each application of a rule by the DPP. Since the unit rule is always applicable it will be applied at most r times. The resolution corresponding to application of the unit rule always has an input clause as one of its parents (see Figure 3.1). Thus the resolution refutation produced will be an input refutation involving at most r resolutions. □

This upper bound represents the best performance that could be expected for a resolution proof. If a set is minimally unsatisfiable, every clause must be a parent for a resolution and so every atom occurring must be annihilated. This means that r resolutions is minimal.

4.3 Renaming

Recall that a renaming of a set of clauses is a set obtained by negating all the occurrences of literals belonging to selected atoms [Meltzer, 1966]. A set is *renamable as a Horn set* if it has a renaming which is a Horn set. Sets that are renamable as Horn sets can be tested for satisfiability by the DPP as efficiently as Horn sets. It is trivial to determine if a set is Horn but not as easy to determine if a set is renamable as a Horn set. There are 2^r possible renamings; if one had to enumerate each one, one could just as well enumerate all truth assignments and test satisfiability. Fortunately, testing whether a set is renamable as a Horn set can be reduced to testing 2-satisfiability (testing satisfiability of sets with at most two literals per clause).^{*} 2-satisfiability is decidable in polynomial time. The problem of renamability as a Horn set is equivalent to the problem of "all but one satisfiability," that is, given a set of clauses, is there a valuation which makes each literal, except possibly one per clause, true? The equivalence is seen by assigning a literal the value True in the valuation iff the literal is negated by the renaming. The problem of "all but one satisfiability" is easily seen to be reducible to 2-satisfiability by forming all clauses of pairs of literals that

^{*} This result was obtained independently by Lewis [Lewis, 1978].

appear in the same clause of the given set. The set derived in this fashion is satisfiable iff the given set is "all but one satisfiable". A formal proof follows.

Theorem 4.6. In polynomial time it can be determined if a set is renamable as a Horn set.

Proof. Given $S = \{C_1, C_2, \dots, C_n\}$, with

$C_i = \{L_{i1}, L_{i2}, L_{i3}, \dots, L_{i, n_i}\}$, $L_{ij} \in \{x_1, \dots, x_r, \bar{x}_1, \dots, \bar{x}_r\}$,

define a new set Q_S , such that Q_S is satisfiable iff S is renamable as a Horn set. Q_S will consist of the following clauses:

$$L_{ij} L_{ik}, \quad 1 \leq j < k \leq n_i, \quad i = 1, \dots, n.$$

Suppose Q_S is satisfiable. Define a renaming of S in which the atoms made True by the valuation are those whose literals are renamed. Suppose that a clause C_i is not Horn after renaming, and that the two positive literals in C_i derive from $L_{ij} L_{ik}$. If, say, L_{ij} , was negative it must have been renamed and so the atom associated with it is given the value True. Hence L_{ij} is given the value False. If L_{ij} was positive, it was not renamed and is therefore given the value False by the valuation. The same argument applies to L_{ik} . $L_{ij} L_{ik}$ is then made False by the valuation, a contradiction. Conversely, given a renaming of S , define a valuation for Q_S by assigning the value True to atoms whose literals are negated and the value False to all others. If a clause $L_{ij} L_{ik}$ in Q_S is False in this valuation, then

clause C_i in S will not be Horn. Thus Q_S is satisfiable iff S is renamable as a Horn set. Since Q_S contains at most nr^2 clauses and because satisfiability of Q_S is decidable in quadratic time, decidability of the renaming of S as a Horn set is polynomial. □

We have demonstrated efficient algorithms for testing the satisfiability of Horn sets as well as sets renamable as Horn sets. Most sets occurring in mathematical contexts will consist predominantly of Horn clauses but will not be Horn or renamable as a Horn set. To test the satisfiability of such a set by the DPP, a natural strategy is to apply the splitting rule to the excess positive literals (those positive literals beyond the one allowed in a Horn clause) of the set (this approach is also suggested in [Schwartz, 1978]). After these splits are made, the DPP will only need to be applied to Horn sets. This strategy corresponds to a proof by cases in a mathematical argument in which one of the hypotheses (or an invoked theorem) is of the form $(A_1 \ \& \ A_2 \ \& \ \dots \ \& \ A_n) \rightarrow (B_1 \ \vee \ B_2)$. When put in CNF this statement becomes the clause $\bar{A}_1 \ \bar{A}_2 \ \dots \ \bar{A}_n \ B_1 B_2$. Splitting on the excess literal, B_2 , corresponds to separately considering the two cases $(A_1 \ \& \ \dots \ \& \ A_n) \rightarrow B_1$ and $(A_1 \ \& \ \dots \ \& \ A_n) \rightarrow B_2$. This procedure will be exponential only in the number of distinct excess positive literals that appear in a set. This leads one to expect good performance in most situations.

Renaming may be applied to reduce the number of distinct excess positive literals. Specifically, given a set S , can we efficiently find the *minimum* k and a set E , with $|E| = k$, so that removal of the atoms in E from S result in a set renamable as a Horn set? This is an open problem, but evidence indicates that the answer is negative. Indeed related problems that are apparently easier can be shown to be NP-hard.

Theorem 4.7. The problem of finding a renaming which minimizes the number of non-Horn clauses is NP-hard.

Proof. We reduce maximum 2-satisfiability [Garey, Johnson and Stockmeyer, 1976] to this minimization problem. The reduction is trivial. Renaming is equivalent to "all but one satisfiability", which, in the case of clauses containing at most 2 literals, is identical to satisfiability. Thus, maximizing the number of clauses that are satisfied by a valuation is identical to maximizing the number of Horn clauses in a renaming. □

Corollary 4.8. Minimizing the number of excess (not necessarily distinct) positive literals by renaming is NP-hard.

Proof. In the special case of 2 literal clauses this is identical to minimizing the number of non-Horn clauses. □

References

- Aho, A. V., J. E. Hopcroft, and J. D. Ullman [1974],
The Design and Analysis of Computer Algorithms,
Addison-Wesley, Reading, MA.
- Bentley, J. L., and M. I. Shamos [1978],
"Divide and conquer linear expected time,"
Inf. Processing Letters 7, 87-91.
- Chang, C. L., and R. C. Lee [1973],
Symbolic Logic and Mechanical Theorem Proving,
Academic Press, New York.
- Church, A. [1956],
Introduction to Mathematical Logic,
Vol. 1, Princeton University Press, Princeton.
- Cook, S. A. [1971],
"The complexity of theorem-proving procedures,"
Proc. 3rd Ann. ACM Symp. on Theory of Computing,
Association for Computing Machinery, New York, 151-158.
- Cook, S. A., and R. A. Reckow [1974],
"On the lengths of proofs in the propositional calculus,"
Proc. 6th Ann. ACM Symp. on Theory of Computing, ACM,
New York, 135-148.
- Davis, H. T. [1962],
The Summation of Series, The Principia Press
of Trinity University, San Antonio, Tx.
- Davis, M. [1963],
"Eliminating the irrelevant from mathematical proofs,"
Proc. Symp. in Applied Math., Vol. 15, 15-30.

- Davis, M., G. Logemann, and D. Loveland [1962],
"A machine program for theorem proving,"
Comm. ACM 5, 394-397.
- Davis, M., and H. Putnam, [1960],
"A computing procedure for quantification theory,"
J. Assoc. Comput. Mach. 7, 201-215.
- Doyle, J., and R. L. Rivest [1976],
"Linear expected time of a simple union-find algorithm,"
Inf. Processing Letters 5, 146-48.
- Dunham, B., and H. Wang [1974],
"Toward feasible solutions of the tautology problem,"
Report RC 4924, Mathematical Sciences Dept., IBM Watson
Research Center, Yorktown Heights, N. Y.
- Erdős, P., and A. Renyi [1959],
"On random graphs, I,"
Publications Mathematicae 6, 290-297.
- Floyd, R. W., and R. L. Rivest [1975],
"Expected time bounds for selection,"
Comm. ACM 18, 165-172.
- Galil, Z. [1977],
"On the complexity of regular resolution and the
Davis-Putnam procedure," *Theoretical Computer Science*
4, 23-46.
- Garey, M. R., and D. S. Johnson [1976],
"Approximation algorithms for combinatorial problems:
an annotated bibliography," in J. F. Traub (ed.),
*Algorithms and Complexity: New Directions and Recent
Results*, Academic Press, New York.

- Garey, M. R., and D. S. Johnson [1979],
Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman, San Francisco.
- Garey, M. R., D. S. Johnson, and L. Stockmeyer [1976],
 "Some simplified NP-complete graph problems,"
Theor. Comput. Sci. 1, 237-267.
- Grimmett, G. R., and C.J.H. McDiarmid [1975],
 "On coloring random graphs," *Math. Proc. Camb. Phil. Soc.* 77, 313-324.
- Guibas, L. J., and E. Szemerédi [1976],
 "The analysis of double hashing,"
Proc. 8th Ann. ACM Symp. on Theory of Computing,
 Association for Computing Machinery, New York, 187-191.
- Henshen L., and L. Wos [1974],
 "Unit refutations and Horn sets,"
J. Assoc. Comp. Mach. 21, 590-605.
- Jordan, C. [1947],
Calculus of Finite Differences, 2nd Ed., Chelsea, New York.
- Karp, R. M. [1972],
 "Reducibility among combinatorial problems,"
 in R. E. Miller and J. W. Thatcher (eds.),
Complexity of Computer Computations, Plenum Press,
 New York, 85-103.
- Karp, R. M. [1976],
 "The probabilistic analysis of some combinatorial search algorithms," in J. F. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*,
 Academic Press, New York, 1-19.

- Klee, V., and G. J. Minty [1972],
"How good is the simplex algorithm?," in O. Shisha
(ed.), *Inequalities III*, Academic Press, New York, 159-75.
- Knuth, D. E. [1973],
The Art of Computer Programming: Sorting and Searching,
Vol. 3, Addison-Wesley, Reading, MA.
- Knuth, D. E., and A. Schönhage [1977],
"The expected linearity of a simple equivalence
algorithm," STAN-CS-77-599, Computer Science Dept.,
Stanford University, Stanford, CA.
- Kowalski, R., and D. Kuehner [1971],
"Linear resolution with selection function,"
Artificial Intelligence 2, 227-260.
- Kozen, D. [1977],
"Lower bounds for natural proof systems,"
Proc. 18th Ann. Symp. on Foundations of Computer Sci.,
IEEE Computer Society, Long Beach, CA, 254-266.
- Ladner, R. E. [1977],
"The computational complexity of provability in systems
of modal propositional logic," *SIAM J. Comput.* 6, 467-480.
- Lewis, H. R. [1978],
"Renaming a set of clauses as a Horn set,"
J. Assoc. Comp. Mach. 25, 134-135.
- Lieberherr, K., and E. Specker [1978],
"Complexity of partial satisfaction,"
unpublished manuscript.

- Loveland, D. W. [1970],
"A linear format for resolution," Symposium on
Automatic Demonstration, *Lecture Notes in Mathematics*
125, Springer-Verlag, Berlin and New York, 147-163.
- Meltzer, B. [1966],
"Theorem-proving for computers: Some results on
resolution and renaming," *Comput. J.* 8, 341-343.
- Perl, Y. [1977],
"Average analysis of simple path algorithms,"
UIUCDCS-R-77-905, Dept. of Computer Science,
University of Illinois at Urbana-Champaign,
Urbana, Illinois.
- Posá, L. [1976],
"Hamilton circuits in random graphs,"
Discrete Mathematics 14, 359-364.
- Reckow, R. A. [1976],
"On the lengths of proofs in the propositional calculus,"
Tech. Report No. 87, Dept. of Computer Science,
University of Toronto, Toronto, Ontario.
- Riordan, J. [1958],
An Introduction to Combinatorial Analysis,
John Wiley & Sons, New York.
- Robinson, J. A. [1965],
"A machine oriented logic based on the resolution
principle," *J. Assoc. Comput. Mach.* 10, 163-174.

- Schaeffer, T. J. [1978],
 "The complexity of satisfiability problems,"
Proc. 10th Ann. ACM Symp. on Theory of Computing,
 Assoc. for Comput. Mach., New York, 216-226.
- Schwartz, J. T. [1978],
 "A survey of program proof technology,"
 Report 001, Dept. of Computer Science, Courant Inst.
 Math. Sci., New York Univ., New York.
- Shostack, R. E. [1976],
 "Refutation graphs," *Artificial Intelligence* 7, 51-64.
- Simon, I. [1971],
 "On the time required by the Davis-Putnam tautology
 recognition algorithm," Report CSRR-2050, Dept. of
 Applied Analysis and Computer Science, University
 of Waterloo, Ontario.
- Spira, P. M. [1973],
 "A new algorithm for finding shortest paths in a graph
 of positive arcs in average time $O(n^2 \log^2 n)$,"
SIAM J. Comput. 2, 28-32.
- Tseitin, G. S. [1968],
 "On the complexity of derivation in propositional
 calculus," in A. O. Slisenko (ed.), *Studies in Constructive
 Mathematics and Mathematical Logic — Part II*,
 Consultants Bureau, New York, 115-125.
- Yao, A. C. [1977],
 "The complexity of pattern matching for a random string,"
 STAN-CS-77-629, Computer Sci. Dept., Stanford Univ., CA.

Yao, A. C., and F. F. Yao [1978],

"On the average case complexity of selecting k-th best,"
Proc. 19th Ann. Symp. on Foundations of Computer Science,
IEEE Computer Society, Long Beach, CA, 280-289.

Yarmush, D. L. [1976],

"The linked conjunct and other algorithms for
mechanical theorem-proving," Report IMM-412,
Courant Inst. Math. Sci., New York Univ., New York.

This book may be kept **SEP 27 1979**

SEP 27 1979

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

OCT 16	
DEC 18 1979	
1980	
DEC 23 1980	
FEB 20 1981	
NOV 30 1981	
FEB 04 1990	
GAYLORD 142	PRINTED IN U.S.A.

NYU NSO-16

c.2

Goldberg

On the complexity of the
Satisfiability problem.

NYU NSO-16

c.2

Goldberg

AUTHOR

On the complexity of the

TITLE

Satisfiability problem.

DATE DUE

BORROWER'S NAME

65.

J. M. Cohen

10/14/1979 RENEWED

10/18/1979 RENEWED

N.Y.U. Courant Institute of
Mathematical Sciences

251 Mercer St.
New York, N. Y. 10012

